Commodore France

CDTV

Developer Notes

CDTV

Developer Notes

The Official CDTV Developer's Notes Issue #1

Welcome to the Commodore CDTV developer community. In order to help you understand and develop products for the Commodore CDTV player we are pulling together various specifications, documents, lists, and resources and sending them to you.

The scope of this project is so broad that there are bound to be mistakes and last second changes. You will also find that in this first batch of materials there is not enough detail to answer specific questions. Other elements may have little relevance to what you are currently working on. We are gathering reference materials as fast as possible and will be sending more detailed information out to you ASAP.

We will be organizing an official CDTV Special Interest Group in the next few weeks as part of the the CATS program for Amiga developers. Commercial developers will receive access to a special section on BIX, a developer manual (when it is written), all developers will receive these developers notes (with weekly updates), invitations to developer conferences, and other privileges, perks, and materials to be determined. If you wish to receive the weekly developer's notes updates and have not already sent us a copy of the Developer Update Registration form please fill it out and return it as soon as you can. In case you have misplaced your copy, another is enclosed.

The first scheduled event, specifically for CDTV developers, will be an informal meeting at the Atlanta Amiga Developers Conference on Thursday June 28, 1990 at 8:00 PM (exact location to be announced at the conference). There will also be CDTV seminars and talks at the conference on Thursday.

*** Work-In-Progress ***

Like CDTV, this developer's pack is in a state of evolution. We will try to keep things as up to date as possible but they may change (actully, you can count on changes) over the next few months. As we get closer to the final relase dates CDTV will get firm. If you need specific information that is not here, have suggestions, comments, and particularly additions that you would like to make please feel free to contact us. You can send your materials via USENET, BIX, or by FAX (no phone calls please.) The numbers are:

USENET - rutgers/uunet!cbmvax!gswright
BIX - gswright
FAX - 603-878-4966

Hopefully, this will grow with your contributions until we have a complete map of the territory for all those who come along later.

Guy Wright
June 8, 1990

Developer Notes Issue #1
Table of Contents

Cover letter
Message from Gail Wellington

Developer Update Registration Form

CDTV Technical Specifications

Amiga
- Amiga Developers Introduction
- Troubleshooting Your Software
- International Keyboard Input
- PAL and NTSC Differences (next issue)
- PAL and Audio
- Amiga Development Guidelines
- Some Caveats to Hardware Level Programmers
- CATS Registration Form
- CATS List of Materials Available

CDTV CD-ROM File Structure
- Commodore Amiga CD-ROM File System
- CD Device Driver Include File
- Rationale for 512 Byte Logical Block Size
- Audio Example Program

User Interface
- User Interface Rules
- TV Specific Imagery
- I/O Issues
- Screen Issues
- Operation of the Machine, Startup and Audio Only
- Application Specific Issues
- New Ideas and Sources for Ideas
- Information System Specific Issues

Developer Tools (next issue)
- Minimum Amiga Configuration
- Video
- Graphics
- CTrac Emulation System for CDTV

CES Report
- CES Press Announcements
- Press Clippings

Commodore
International Limited

Executive Offices
1200 Wilson Drive
West Chester, PA 19380
(215) 431-9100

June 13, 1990

Dear Developer:

Whew! CES was quite an event for the CDTV team! We graduated from the trailer in January to inside the hall in June, albeit in a back room. Our schedule was overwhelmed with visitors. All the prototype units on display worked well throughout the entire four days despite running continuously in a very warm area.

Special thanks to all of you who worked so hard to provide us with demo disks. Also, bouquets to Peter Black of Xiphias, Steve Buerkle of Applied Optical Media, Laura Buddine of Tiger Media, and Chuck Bradley and Gordon Clarke of mediaMAXX who did presentations in the demo area. The back room didn't prevent us from making our presence felt.

All reactions indicate a rosy future for our "baby" when it is launched in the autumn. But we all have lots to do to make the promise a reality. This package contains information that will help you do that. We are also asking for your input, particularly as it relates to the User Interface Specifications. It will be essential that products for CDTV be easy to use. One way to assist in this is to have a standard user interface. A first proposal is included in this document. We ask you to review it, to give us your comments, but especially to try it and give us end-user feedback. This will help us evolve and establish a practical standard.

Thanks for supporting Commodore's CDTV with your applications.

Sincerely,

Gail P. Wellington
Director, Special Projects

Attachments
GPW:LC

DEVELOPERS' UPDATE REGISTRATION
COMMODORE INTERNATIONAL LIMITED
SPECIAL PROJECTS GROUP
THIRD-PARTY PROJECT OUTLINE

Company Name _____

Address _____

_____

_____

Phone _____

Fax _____

Technical Contact _____

Marketing Contact _____

Product Title _____

Product Type _____

Target Market _____

Probable RRP _____

Brief Description: _____

_____

_____

PAL _____ NTSC _____

Demonstrable by (check one):

Sept 1, 1990 _____

Dec 31, 1990 _____

Est. release date _____

Languages Available & Dates:

English _____
French _____
Italian _____
German _____
Spanish _____
Other _____

One product per page. Information will be held in confidence if you check here _____.

Return to
Gail Wellington
Commodore
1200 Wilson Drive
West Chester, PA 19380
fax (215) 431-9156

June 2, 1990

WARNING - The information contained herein is subject to change without notice. Commodore specifically does not make any endorsement or representation with respect to use, results, or performance of the information (including without limitation its capabilities, appropriateness, reliability, currentness or availability.)

DISCLAIMER - This information is provided "as is" without warranty of any kind, either express or implied. The entire risk as to the use of this information is assummed by the user. In no event will Commoodore or its affiliated companies be liable for any damages, direct, incidental, special or consequential, resulting from any defect in the information, even if advised of the possibilities of such damages.

June 2, 1990

General Specifications

Working name: Baby
Final name: CDTV

System overview:

An interactive consumer product cosmetically resembling a VCR, intended for the home entertainment environment. Easy to use via infrared remote device (no keyboard required.) Based on Amiga technology, it consists of CD ROM drive (with CD Audio capability) real time clock, and remote controller. It can be easily connected to a TV or monitor and stereo system. It is targeted toward the family and the audio and entertainment enthusiast.

Target Market:

Educated adults, ages 24-49, who are interested in their own development and in providing their families with the greatest opportunity for advancement. This product is the multi-media device that allows them to use their TVs in a more intelligent and interactive manner by giving easy access to unlimited educational, informational and entertainment-oriented applications using the mutimedia features of graphics and sound.

Major Functions:

- Playing Amiga CD ROM software which can integrate various qualities of audio
- Playing CD quality audio while displaying graphics (CD+G)
- Playing standard high-fidelity CD-audio disks
- Can be expanded to an Amiga computer
- CD midi-in/midi-out

Technical Specifications

Central Processing Unit:

- MC68000 (Motorola) 16/32 bit main CPU

CPU speed:

- 7.15909 MHz (NTSC)
- 7.09379 MHz (PAL) [NOTE: CPU speeds can vary when using an external system clock such as with genlocking units.]

Custom Chips:
- 3 Amiga specific custom chips (Agnus, Paula, and Denise) which enhance system performance by taking over tasks such as handling video, sound, direct memory access (DMA), or graphics.

Other Custom Chips:
- CDTV will also have other custom chips for handling the CD-ROM interface and IR control.

Memory:
- 1Mb (megabyte) chip RAM
- 2K non-volatile RAM reserved for system (clock, prefs, etc.)
- 512K ROM

Internal Slots:
- Intelligent video slot (for optional genlock, RF board, etc.) 15 pin edge connector
- DMA slot - for SCII, LAN, etc.

Video outputs:
- Analog RGB, Digital RGB (DB-23 connector)
- Composite video NTSC or PAL (RCA connector)
- Component video Y-C (S connector type for S-VHS and Hi8)
- RF Modulated (F connector)
- Optional genlock capabilities via plug-in module. Three-mode (CD,video source or mixed) under software control.

Video Display (General):
- 400 lines/vertical frequency 60Hz (NTSC)
- 512 lines/vertical frequency 50Hz (PAL)
- graphic co-processor with beam synced draw, fill, and move modes (blitter)
- Maximum 1 Mb video memory (chip memory)
- Palette of 4096 colors
- Maximum 6 bitplanes
- 8 sprites per scanline

Text Modes:
- 80 characters/25 lines
- 60 characters/25 lines
- Various font sizes and types selectable
- Screen colors user definable

Graphics Modes:
(NTSC)
- 320 X 200 non-interlaced 32 colors
- 640 X 200 non-interlaced 16 colors
- 320 X 400 interlaced 32 colors
- 640 X 400 interlaced 16 colors
- plus overscan and HAM modes

(PAL)
- 320 X 256 non-interlaced 32 colours
- 640 X 256 non-interlaced 16 colours
- 320 X 512 interlaced 32 colours
- 640 X 512 interlaced 16 colours
- plus overscan, HAM, etc.

CD-ROM Drive Specs:
- Sony/Philips type CD-ROM standard mode 1, Mode 2
- Data readout from disk : 153 KBytes/sec (mode 1)
  : 171 KBytes/sec (mode 2)
  : 2 Megabytes/sec (burst)
- Average access time : 0.5 sec
- Maximum access time : 0.8 sec
- Soft read error : Less than 10e-9
- Hard read error : Less than 10e-12
- Seek error : Less than 10e-6
- Commands : CD-ROM, CD-Audio, CD+G
- MTBF : 10,000 P.O.H.
- Standard supported : ISO-9660
- Data Capacity : 540 MegaBytes (aprox)
  (about 700 Amiga floppy disks)

CD Audio Specs:
- 8X oversampling
- Audio output : External 1.4VRMS, 10K OHM
- Frequency response : 20-20KHz
- Signal/Noise : -75DB (Typical)
- Channel Separation : -75DB (Typical)
- Harmonic Distortion : 0.08% at 1 KHz
- Dual 16 bit D/A converter plus 64 levels of attenuation
- Maximum audio capacity : 14 hours - AM quality -

Sample Rates : variable from CD Audio rate (44Khz)
to 6Khz

Computer Generated Audio Specs:
- 4 independent sound channels (right and left)
- Complex waveforms
- Sound buffer up to 128K, unlimited seamless sound
- 8-Bit D/A converter plus 6-Bit volume control

Audio Extras:
- Capable of sound mapping Amiga over CD-DA sound

Rear Ports:

- Centronics Parallel interface
- RS-232 Serial interface
- External floppy disk drive interface (Amiga floppy disk drive compatible)
- Hardwired alternative to IR for keyboard, mouse, joystick
- 2 audio output ports (RCA type plug) (requires external audio amplifier.)
- Midi-in/midi-out

Front Port:

- Stereo Headphone jack
- Port for optional Personal RAM Card (up to 64K)

Power Consumption:

- 50W (Average AC100-240V, 50/60Hz)

Environmental Conditions:

- Operating Temperature : 5 C to 40 C (non-condensing)
  41 F to 104 F
- Operating Humidity : 25% to 80% (non-condensing)

Front Panel Display:

- Fluorescent (white characters on black)
- Time, track, and volume level
- Clock remains lit when unit is turned off
- Display controllable by application

Front Panel Controls:

- Power On/Off
- Headphone volume Up/Down
- Play/Pause
- Stop
- Forward/Reverse - Scan/Skip
- CD/TV
- Reset

IR (Infrared) Remote Unit Specs:

- Proprietary high-speed IR signal
- Power supplied by 2 AA batteries
- 10 function keys plus Shift key (20 total)
- Up, Down, Left, Right movement button
- Two select keys
- CD audio - Reverse, Forward, Play/Pause, Headphone Volume and Stop keys
- Computer reset function

Optional Accessories:

- External floppy disk drive
- Trackball
- Joystick (third party)
- keyboard
- MIDI in, out, through (third party)
- Personal RAM or ROM card
- Genlock
- Expansion module to house hard disk drive, modem, floppy disk drive
- Keyboard IR interface with track ball
- Keyboard
- Two player IR interface

Added Raw Keyboard Codes:

- 6C Play/Pause
- 6D Stop
- 6E Fast Forward
- 6F Fast Reverse

Amiga

CDTV Developer Notes Issue #1
6/8/90

Amiga

This section covers some of the basics of the Amiga computer, some specific Amiga programming dos and don'ts, and material relating to multi-language, multi-format development considerations. Much of this material was gathered from CATS (Commodore Amiga Technical Support) materials. We have also included a CATS registration form and list of materials available from CATS. We strongly recommend that CDTV developers register in the CATS program.

*** NOTE ***

UNLIKE other sections in this first issue of developer notes, this material is REAL. While a few details may change (CDTV hardware will be slightly different than a stock Amiga) the sections on Amiga programming dos and don'ts WILL NOT CHANGE!

Although this material is principally for developers new to the Amiga, all developers would benefit from re-reading it, especially the sections following the introduction.

Amiga Developers Introduction

The Amiga family of computers consists of several models, each of which hasbeen designed on the same premise——to provide the user with a low costcomputer that features high cost performance. The Amiga does this through the use of custom silicon hardware that yields advanced graphics and soundfeatures.

There are three distinct models that make up the Amiga computer family: theA500, A1000, and A2000. Though the models differ in price and features, they have a common hardware nucleus that makes them software compatible with one another. This chapter describes the Amiga's hardware components andgives a brief overview of its graphics and sound features.

Components of the Amiga

These are the hardware components of the Amiga:

Motorola MC68000 16/32 bit main processor. The Amiga also supports the 68010, 68020, and 68030 processors as an option.

512K bytes of internal RAM, expandable to 1 MB on the A500 andA2000.

256K bytes of ROM containing a real time, multitasking operating system with sound, graphics, and animation support routines.

Built-in 3.5 inch double sided disk drive.

Expansion disk port for connecting up to three additional diskdrives, which may be either 3.5 inch or 5.25 inch, double sided.

Fully programmable RS-232-C serial port.

Fully programmable parallel port.

Two button opto-mechanical mouse.

Two reconfigurable controller ports (for mice, joysticks, light pens,paddles, or custom controllers).

A professional keyboard with numeric keypad, 10 function keys, andcursor keys. A variety of international keyboards are alsosupported.

Ports for simultaneous composite video, and analog or digital RGBoutput.

Ports for left and right stereo audio from four special purposeaudio channels.

Expansion options that allow you to add RAM, additional diskdrives (floppy or hard), peripherals, or coprocessors.

THE MC68000 AND THE AMIGA CUSTOM CHIPS

The Motorola 68000 is a 16/32 bit microprocessor. The system clock speed for NTSC Amigas is 7.15909 megahertz (PAL, 7.09379 MHz). These speeds may vary when using an external system clock, such as from a genlock. The 68000 has an address space of 16 megabytes. In the Amiga, the 68000 can address over 8 megabytes of contiguous random access memory (RAM).

In addition to the 68000, the Amiga contains special purpose hardware known as the "custom chips" that greatly enhance system performance. The term "custom chips" refers to the 3 integrated circuits which were designed specifically for the Amiga computer. These three custom chips (called Agnus, Paula, and Denise) each contain the logic to handle a specific set of tasks, such as video, sound, direct memory access (DMA), or graphics.

Among other functions, the custom chips provide the following:

Bitplane generated, high resolution graphics capable of supporting both PAL and NTSC video standards.

On NTSC systems the Amiga typically produces a 320 by 200 non-interlaced or 320 by 400 interlaced display in 32 colors and a 640 by 200 non-interlaced or 640 by 400 interlaced display in 16 colors.

On PAL systems, the Amiga typically produces a 320 by 256 non-interlaced or 320 by 512 interlaced display in 32 colors, and a 640 by 256 non-interlaced or 640 by 512 interlaced display in 16 colors.

Additional video modes allow for the display of up to 4,096 colors on screen simultaneously (hold-and-modify) or provide for larger, higher resolution displays (overscan).

A custom display coprocessor that allows changes to most of the special purpose registers in synchronization with the position of the video beam. This allows such special effects as mid-screen changes to the color palette, splitting the screen into multiple horizontal slices each having different video resolutions and color depths, beam synchronized interrupt generation for the 68000 and more. The coprocessor can trigger many times per screen, in the middle of lines, and at the beginning or during the blanking interval. The coprocessor itself can directly affect most of the registers in the other custom chips, freeing the 68000 for general computing tasks.

32 system color registers, each of which contains a twelve bit number as four bits of RED, four bits of GREEN, and four bits of BLUE intensity information. This allows a system color palette of 4,096 different choices of color for each register.

Eight reusable 16 bit wide sprites with up to 15 color choices per sprite pixel (when sprites are paired). A sprite is an easily movable graphics object whose display is entirely independent of the background (called a playfield); sprites can be displayed over or under this background. A sprite is 16 low resolution pixels wide and an arbitrary number of

lines tall. After producing the last line of a sprite on the screen, a sprite DMA channel may be used to produce yet another sprite image elsewhere on screen (with at least one horizontal line between each reuse of a sprite processor). Thus, many small sprites can be produced by simply reusing the sprite processors appropriately.

Dynamically controllable inter-object priority, with collision detection. This means that the system can dynamically control the video priority between the sprite objects and the bitplane backgrounds (playfields). You can control which object or objects appear over or under the background at any time. Additionally, you can use system hardware to detect collisions between objects and have your program react to such collisions.

Custom bit blitter used for high speed data movement, adaptable to bitplane animation. The blitter has been designed to efficiently retrieve data from up to three sources, combine the data in one of 256 different possible ways, and optionally store the combined data in a destination area. This is one of the situations where the 68000 gives up memory cycles to a DMA channel that can do the job more efficiently (see below). The bit blitter, in a special mode, draws patterned lines into rectangularly organized memory regions at a speed of about 1 million dots per second; and it can efficiently handle area fill.

Audio consisting of four digital channels with independently programmable volume and sampling rate. The audio channels retrieve their control and data via direct memory access. Once started, each channel can automatically play a specified waveform without further processor interaction. Two channels are directed into each of the two stereo audio outputs. The audio channels may be linked together to provide amplitude or frequency modulation or both forms of modulation simultaneously.

DMA controlled floppy disk read and write on a full track basis. This means that the built-in disk can read over 5600 bytes of data in a single disk revolution (11 sectors of 512 bytes each).

The internal memory shared by the custom chips and the 68000 CPU is also called "chip memory". The original custom chips in the Amiga were designed to be able to physically access up to 512K bytes of shared memory. The new version of the Agnus custom chip was created which allows the graphics and audio hardware to access up to a full megabyte of memory.

The Amiga 500 and 2000 models were designed to be able to accept the new Agnus custom chip, called "Fat Agnus," due to its square shape. Hence, the A500 and A2000 have allocated a chip memory space of 1 MB. This entire 1 MB space is subject to the arbitration logic that controls the CPU and custom chip accesses. On the A1000, only the first 512K bytes of memory space is shared, chip memory.

These custom chips and the 68000 share memory on a fully interleaved basis. Since the 68000 only needs to access the memory bus during each alternate clock cycle in order to run full speed, the rest of the time the memory bus is free for other activities. The custom chips use the memory bus during these free cycles, effectively allowing the 68000 to run at full rated speed

most of the time. We say "most of the time" because there are some occasions when the special purpose hardware steals memory cycles from the 68000, but with good reason. Specifically, the coprocessor and the data moving DMA channel called the blitter can each steal time from the 68000 for jobs they can do better than the 68000. Thus, the system DMA channels are designed with maximum performance in mind. The job to be done is performed by the most efficient hardware element available. Even when such cycle stealing occurs, it only blocks the 68000's access to the internal, shared memory. When using ROM or external memory, the 68000 always runs at full speed.

Another primary feature of the Amiga hardware is the ability to dynamically control which part of the chip memory is used for the background display, audio, and sprites. The Amiga is not limited to a small, specific area of RAM for a frame buffer. Instead, the system allows display bitplanes, sprite processor control lists, coprocessor instruction lists, or audio channel control lists to be located anywhere within chip memory.

This same region of memory can be accessed by the bit blitter. This means, for example, that the user can store partial images at scattered areas of chip memory and use these images for animation effects by rapidly replacing on screen material while saving and restoring background images.In fact, the Amiga includes firmware support for display definition and control as well as support for animated objects embedded within playfields.

VCR AND DIRECT CAMERA INTERFACE

In addition to the connectors for monochrome composite, and analog or digital RGB monitors, the Amiga can be expanded to include a VCR or camera interface. This system is capable of synchronizing with an external video source and replacing the system background color with the external image. This allows development of fully integrated video images with computer generated graphics. Laser disk input is accepted in the same manner.

PERIPHERALS

Floppy disk storage is provided by a built in, 3.5 inch floppy disk drive. Disks are 80 track, double sided, and formatted as 11 sectors per track, 512 bytes per sector (over 900,000 bytes per disk). The disk controller can read and write 320/360K IBM PC*(Tm (MS-DOS*(Tm) formatted 3.5 or 5.25 inch disks, and 640/720K IBM PC (MS-DOS) formatted 3.5 inch disks. External 3.5 inch or 5.25 inch disk drives can be added to the system through the expansion connector.

Circuitry for some of the peripherals resides on Paula. Other chips handle various signals not specifically assigned to any of the custom chips, including modem controls, disk status sensing, disk motor and stepping controls, ROM enable, parallel input/output interface, and keyboard interface.

The Amiga includes a standard RS-232-C serial port for external serial input/output devices.

A keyboard with numeric keypad, cursor controls and 10 function keys is included in the base system. For maximum flexibility, both key-down and key-up signals are sent. The Amiga also supports a variety of international keyboards. Many other types of controllers can be attached through the two controller ports on the base unit. You can use a mouse, joystick, keypad, trackball, light pen, or steering wheel controller in either of the controller ports.

SYSTEM EXPANDABILITY AND ADAPTABILITY

New peripheral devices may be easily added to all Amiga models.
These devices are automatically recognized and used by system software through a well defined, well documented linking procedure called AUTOCONFIGTm.

On the A500 and A1000 models, peripheral devices can be added to the Amiga's 86 pin expansion connector, including additional external RAM.
Extra disk units may be added from a connector at the rear of the unit.

The A2000 model provides the user with the same features as the A500 or A1000, but with the added convenience of simple and extensive expandability.
The 86 pin, external connector of the A1000 and A500 is not externally accessible on the A2000. Instead, the A2000 contains 7 internal slots that allow many types of expansion boards to be quickly and easily added inside the machine. These expansion boards may contain coprocessors, RAM expansion, hard disk controllers, video or I/O ports. There is also room to mount both floppy and hard disks internally. The A2000 also supports the special Bridgeboard*(Tm coprocessor card. This provides a complete IBM PC*(Tm on a card and allows the Amiga to run MS-DOS*(Tm compatible software, while simultaneously running native Amiga software.

As a general rule, it is not permissible to directly access the hardware in the Amiga unless your software either has full control of the system, or has arbitrated via the OS for exclusive access to the particular parts of the hardware you wish to control.

Almost all of the hardware discussed in the Amiga Hardware Reference Manual, most notably the Blitter, Copper, playfield, sprite, CIA, trackdisk, and system control hardware, are in either exclusive or arbitrated use by portions of the Amiga OS in any running Amiga system. Additional hardware, such as the audio, parallel, and serial hardware, may be in use by applications which have allocated their use through the system software.

Before attempting to directly manipulate any part of the hardware in the Amiga's multitasking environment, your application must first be granted exclusive access to that hardware by the operating system library, device, or resource which arbitrates its ownership. The operating system functions for requesting and receiving control of parts of the Amiga hardware are varied and are not within the scope of this manual. Generally such functions, when available, will be found in the library, device, or resource which manages that portion of the Amiga hardware in the multitasking environment. The following list will help you to find the appropriate operating system functions or mechanisms which may exist for arbitrated access to the hardware discussed in this manual.

Copper, Playfield, Sprite, Blitter - graphics.library
Audio - audio.device
Trackdisk - trackdisk.device, disk.resource
Serial - serial.device, misc.resource
Parallel - parallel.device, cia.resource, misc.resource
Gameport - input.device, gameport.device, potgo.resource
Keyboard - input.device, keyboard.device
System Control - graphics.library, exec.library (interrupts)

The Amiga hardware include files are generally supplied with your compiler or assembler. Listings of the hardware include files may also be found in the Addison-Wesley Amiga ROM Kernel Manual "Includes and Autodocs". Generally, the include file label names are very similar to the equivalent hardware register list names with the following typical differences.

Address registers which have low word and high word components are generally listed as two word sized registers in the hardware register list, with each register name containing either a suffix or embedded "L" or "H" for low and high. The include file label for the same register will generally treat the whole register as a longword (32 bit) register, and therefore will not contain the "L" or "H" distinction.

Related sequential registers which are given individual names with number suffixes in the hardware register list, are generally referenced from a single base register definition in the include files. For example, the color registers in the hardware list (COLOR00, COLOR01, etc.) would be referenced from the "color" label defined in "hardware/custom.i" (color+0, color+2, etc.).

Examples of how to define the correct register offset can be found in the hw_examples.i file listed in Appendix J in the Amiga Hardware Reference Manual.

Some Caveats to Hardware Level Programmers

The Amiga is available in a variety of models and configurations, and is further diversified by a wealth of add-on expansion peripherals and processor replacements. In addition, even standard Amiga hardware such as the keyboard and floppy disks, are supplied by a number of different manufacturers and may vary subtly in both their timing and in their ability to perform outside of their specified capabilities.

The Amiga operating system is designed to operate the Amiga hardware within spec, adapt to different hardware and RAM configurations, and generally provide upward compatibility with any future hardware upgrades or "add ons" envisioned by the designers. For maximum upward compatibility, it is strongly suggested that programmers deal with the hardware through the commands and functions provided by the Amiga operating system.

If you find it necessary to program the hardware directly, then it is your responsibility to write code which will work properly on various models and configurations. Be sure to properly request and gain control of the hardware you are manipulating, and be especially careful in the following areas:

Do not jump into ROM. Beware of any example code that calls routines in the $F80000 to $FFFFFF range. These are ROM addresses and the ROM routines WILL move with every OS revision. The only supported interface to system ROM code is through the provided library, device, and resource calls.

Do not modify or depend on the format of any private system structures.

This includes the poking of copper lists, memory lists, and library bases.

Do not depend on any address containing any particular system structure or type of memory. The system modules dynamically allocate their memory space when they are initialized. The addresses of system structures and buffers differ with every OS, every model, and every configuration, as does the amount of free memory and system stack usage. Remember that all data for direct custom chip access must be in CHIP RAM. This includes bit images (bitplanes, sprites, etc), sound samples, trackdisk buffers, and copper lists.

Do not write spurious data to, or interpret undefined data from currently unused bits or addresses in the custom chip space. All undefined bits must be set to zero for writes, and ignored on reads.

Do not write data past the current end of custom chip space. Custom chips may be extended or enhanced to provide additional registers, or to use currently undefined bits in existing registers.

All custom chip registers are read only OR write only. Do not read write only registers, and do not write to read only registers.

Do not read, write, or use any currently undefined address ranges. The current and future usage of such areas is reserved by Commodore and is definitely subject to change.

If you are using the system libraries, devices, and resources, you must follow the defined interface. Assembler programmers (and compiler writers) must enter functions through the library base jump tables, with arguments passed as longs and library base address in A6. Results returned in D0 must be tested, and the contents of D0-D1/A0-A1 must be assumed gone after a system call.

The assembler TAS instruction should not be used in any Amiga program. The TAS instruction assumes an indivisible read-modify-write but this can be defeated by system DMA. Instead use BSET and BCLR. These instructions perform a test and set operation which cannot be interrupted.

TAS is only needed for a multiple CPU system. On a single CPU system, the BSET and BCLR instructions are identical to TAS, as the 68000 does not interrupt instructions in the middle. BSET and BCLR first test, then set bits.

Do not use assembler instructions which are privileged on any 68000 family processor, most notably MOVE SR,‹ ea› which is privileged on the 68010/20/30. Use the Exec function GetCC() instead of MOVE SR, or use the appropriate non-privileged instruction as shown below:

CPU	User Mode	Super Mode
68000	MOVE SR, ‹ ea›	MOVE SR,‹ ea›
68010/20/30	MOVE CCR, ‹ ea›	MOVE SR,‹ ea›

All addresses must be 32 bits. Do not use the upper 8 bits for other data, and do not use signed variables or signed math for addresses. Do not execute code on your stack or use self-modifying code since such code can be defeated by the caching capabilities of some 68xxx processors. And never use processor or clock speed dependent software loops for timing delays. See Appendix F of the Amiga Hardware Reference Manual for information on using an 8520 timer for delays.

When strobing any register which responds to either a read or a write, (for example copjmp2) be sure to use a MOVE.W #$00, not CLR.W. The CLR instruction causes a read and a clear (two accesses) on a 68000, but only a single access on 68020 and above. This will give different results on different processors.

If you are programming at the hardware level, you must follow hardware interfacing specifications. All hardware is NOT the same. Do not assume that low level hacks for speed or copy protection will work on all drives, or all keyboards, or all systems, or future systems. Test your software on many different systems, with different processors, OS, hardware, and RAM configurations.

Troubleshooting Your Software

by Carolyn Scheppner

Many Amiga programming errors have classic symptoms. This guide gives some tips to help you find and eliminate these problems in your software.

CLI Error Messages

This is caused by calling exit(n) with an invalid or missing return value. Assembler programmers using startup code should jump to the startup code's _exit with a valid return value on the stack. Programs without startup code should return with a valid value in D0. Valid return values are defined in *libraries/dos.h* and *i*. Other values (-1 for instance) can cause CLI error messages such as "not an object module".

CLI Won't Close on RUN

A CLI can't close if a program has a lock on the CLI input or output stream ("*"). If your program is RUN >NIL: from a CLI, that CLI should be able to close unless your code or your compiler's startup code explicitly opens "*".

Crashes and Memory Corruption at Run Time

Memory corruption, address errors, and illegal instruction errors are generally caused by wild pointers - pointers which are uninitialized, incorrectly initialized, or point to memory or resources which have already been freed or closed. You may be accidently modifying or incrementing a pointer later used to free memory or close a resource. The pointer may be one that you use directly, or indirectly, such as in a structure which is passed to a system routine.

Amiga functions which open system resources or allocate memory typically return a pointer; if the open or allocate fails, zero is usually returned. So you must test the return value of a system call for success before using it as a pointer.

Test utilities such as *MemWatch* and *MemMung* can help catch the use of uninitialized pointers or freed memory. *MemMung* is a torture test which sets freed memory areas and location $0 to odd values (if your program is written correctly, this will have no effect; if not, your program is more likely to fail). *MemWatch* is a watchdog utility which reports modification of low memory (wild pointers often point to low memory, especially $0).

Memory corruption and crashes can also be caused by calling functions with the wrong arguments or missing arguments (for example SetAPen(3) or SetAPen(win,3), instead of SetAPen(rp,3)). Another possibility is that you might be overflowing your stack. The compiler's stack checking option may be able to catch this (with Lattice, -v disables stack checking). Cut stack usage by dynamically allocating large structures, buffers, and arrays if they are currently defined inside main() or your other functions. If you are using short integers be sure to explicitly type any long constants (e.g. 42L). For example, with short integers, the expression 1 << 17 may become zero. If corruption is occurring during exit, use printf (or kprintf, etc.) with Delay(n) to slow down your cleanup and broadcast each step.

A bad pointer which causes a system crash will often be reported as a guru meditation 00000003 or 4. Numbers in the range 00000006 - B may also indicate a problem with pointers. These numbers correspond to the hardware defined CPU exceptions of Motorola's 68000-family of processors. Generally these occur when the CPU tries to access a non-existent memory location or execute an illegal instruction. Other guru meditation numbers are Amiga-specific, but may be caused by wild pointers; for the meaning of these codes, refer to the include file *exec/alerts.h*.

Crashes - After Exit

If your program crashes after exiting only when your program is started from the Workbench, then you are probably UnLock()ing one of the WBStartup message wa_Locks, or UnLock()ing the Lock returned from an initial CurrentDir() call. If you call CurrentDir() in your application, you should save the first lock it returns and then call CurrentDir() on that lock before you exit.

If you are crashing from both Workbench and CLI and you are only crashing *after* exit, then you may be freeing or closing something twice. Also, you may be freeing or closing something that you did not allocate or open.

A crash after your program exits can also be caused by leaving an outstanding device IO request or other wakeup request. If you send an IO request and then exit, Exec, upon completion of that IO request, will send a reply message to a port that no longer exists. You must abort and then WaitIO() on any pending IO requests before you free things and exit. See the autodocs for your device and for Exec AbortIO() and WaitIO(). Similar problems can be caused by deleting a subtask that is in a wait loop such as WaitTOF(). Only delete subtasks when you are sure they are in a safe state such as Wait(0L).

Crashes - Subtasks, Interrupts

If part of your code runs on a different stack or on the system stack, you must turn off compiler stack-checking options (Lattice uses the -v flag to disable stack checking). If part of your code is called directly by the system or by other tasks, you must use the large code / large data model, or use compiler functions or options to assure that the correct base registers are set up for your subtask or interrupt code.

Crashes - Window Related

Be careful not to call CloseWindow() during a while(msg=GetMsg(...)) loop on that window's port because the next GetMsg() will be on a freed pointer. Also, use ModifyIDCMP(NULL) with care, especially if you are using one port with multiple windows. Be sure to ClearMenuStrip() any menus before closing a window, and do not free items such as dynamically allocated gadgets and menus while they are attached to a window.

Crashes - Workbench Only

If you are crashing near the first DOS call, either your stack is too small or your startup code does not GetMsg() the WBStartup message from the process message port. If your program crashes only when started from Workbench and your startup code opens no stdio window or NIL: file handles for Workbench programs, then make sure you are not writing anything to stdout (e.g. printf()) when started from Workbench (argc==0). See also "Crashes - After Exit" above.

Disk Icon Won't Go Away

This occurs when a program leaves a lock on one or more of a disk's files or directories.

Fails Only On the 68020/30

In general this occurs whenever an application inadvertently contains a CPU dependency. The following programming practices will lead to programs which fail on the Motorola® 68020/030 (but run OK on the 68000):

- Using the upper byte of addresses for flags.
- Doing signed math on addresses.
- Writing self-modifying code.
- Using the MOVE SR assembler instruction (use Exec GetCC() instead).
- Using software delay or timing loops.
- Making assumptions about the order in which asynchronous tasks will finish.

Special features of the 68020/30 processors can also cause problems for programs written on the 68000. For example, an invalid cache entry due to DMA or other non-processor modification of data which has already been cached; a different exception stack frame; interrupt auto-vectors moved by VBR; the 68020/30 CLR instruction which does a single write access unlike the 68000 CLR instruction which does a separate read and write access (this might effect a read-triggered register in IO space - use MOVE instead).

Fails Only On the 68000

Again, a program which fails only on certain processors contains a CPU dependency. The following programming practices can cause this problem:

- Software delay loops.
- Word or longword access of an odd address (illegal on the 68000).
- Assumptions about the order in which asynchronous tasks will finish.
- Using compiler flags which have generated inline 68881/68882 math coprocessor instructions or 68020/30 specific code.
- Using the CLR instruction on a hardware register (it's behavior on the 68000 differs from the 68020/030 (use MOVE instead)).

Fails Only on Older ROMs or Older Workbench

This can be caused by calling functions or using structures which do not exist in the older versions of the operating system. Or you may be asking for a library version higher than you need. Ask for the lowest version which provides the functions your application

requires (usually 33). You should *not* use the `#define LIBRARY_VERSION` from the include files when you open a library. Also make sure you check OpenLibrary() calls for success (a non-zero return value). If the library you request is not available, exit gracefully and informatively.

Fails only on Newer ROMs or Newer Workbench

This should not happen with proper programming. Possible causes are:

- Running too close to your stack limits or the memory limits of a base machine (newer versions of the operating system may use slightly more stack in system calls, and usually use more free memory).
- Using system functions improperly.
- Not testing function return values.
- Using improperly initialized pointers
- Assuming that a system variable (such as a Flags field) is B if it is not A.
- Failing to initialize formerly reserved structure fields to zero.
- Violating Amiga programming guidelines (for example: depending on or poking private system structures, jumping into ROM, depending on undocumented or unsupported behaviors).
- Failing to read the function autodocs.

Fails On CHIP-RAM-Only Machines

This is caused by specifically asking for or requiring `MEMF_FAST` memory. If you don't need chip memory, ask for memory type 0L, or `MEMF_CLEAR`, or `MEMF_PUBLIC|MEMF_CLEAR` as applicable. If there is fast memory available, you will be given fast memory. If not, you will get chip memory.

Fails Only on Machines with FAST RAM

Data and buffers which will be accessed directly by the custom chips *must* be in chip memory. This includes bitplanes (use OpenScreen() or AllocRaster()), audio samples, trackdisk buffers, and the graphic image data for sprites, pointers, bobs, images, gadgets, etc. Use compiler or linker flags to force chip memory loading of any initialized data that needs to be in chip memory. You could also dynamically allocate chip memory and copy the initialized data there.

Fails Only with Enhanced Chips

This is usually caused by writing or reading addresses past the end of register space on older custom chips, or writing a non-zero value to bits which are undefined in older chip registers, or failing to mask out undefined bits when interpreting the value read from a chip register.

Fireworks

A dazzling pyrotechnic video display is caused by trashing or freeing a copper list which is in use, or trashing the pointers to the copper list. If you aren't messing with copper lists, see "Crashes and Memory Corruption".

Graphics - Corrupted Images

The bit data for graphic images such as sprites, pointers, bobs, and gadgets *must* be in chip memory. Check your compiler manual for directives or flags which will place your graphic image data in chip memory. Alternately you could allocate chip memory and copy the graphic image there.

Hang - Single Program Only

Program hangs are generally caused by Wait()ing on the wrong signal bits, on the wrong port, on the wrong message, or on some other event that will never occur. They can also be caused by verify deadlocks. Be sure to turn off all Intuition `VERIFY` messages (such as `MENUVERIFY`) before calling AutoRequest() or doing disk access.

Hang - Whole System

This is generally caused by a Disable() without a corresponding Enable(). It can also be caused by memory corruption, especially corruption of low memory. See "Crashes and Memory Corruption" above.

Memory Loss

First, make sure that your program is actually causing the memory loss. Boot with a normal Workbench disk whose *s:startup-sequence LoadWB* command line has been changed to *LoadWB -debug*. It is important to boot with a standard Workbench because some third party applications such as background utilities, shells, and network handlers dynamically allocate and free memory. Arrange all windows so that part of the

Workbench backdrop window is accessible and so that no window rearrangement will be needed to run your program. Select flushlibs from the rightmost Workbench menu. Any disk-loaded fonts, libraries, devices, etc. that are not currently open will be flushed from memory. Wait a few seconds, then click on the Workbench backdrop. Write down the amount of free memory displayed in the Workbench title bar. Now without rearranging any windows, run your program and use all of the program features. Exit your program, wait a few seconds, then click on the Workbench backdrop. Now select flushlibs, wait a few seconds and write down this final free amount. If this matches the first value you wrote down, then your program is fine and is not causing a memory loss.

If memory was actually lost and your program can be run from CLI or Workbench, then try the above procedure with both methods of starting your program. See "Memory Loss - CLI Only" and "Memory Loss - Workbench Only" as appropriate.

If you lose memory from both Workbench and CLI, then make sure all calls to functions which open/allocate/create/lock have a matching call to the corresponding close/free/delete/unlock function (there are a few system calls that do not require a corresponding free - check the autodocs). Generally, the close/free/delete/unlock calls should be in the opposite order of the allocations.

If you are losing a small, fixed amount of memory, look for a structure of that size in the Structure Offsets listing in the *Includes and Autodocs* manual. For example, a loss of exactly 24 bytes is probably a Lock which has not been UnLock()ed. If you are using ScrollRaster(), be aware that ScrollRaster() left or right in a `SUPERBITMAP` window with no TmpRas will currently lose memory (workaround - attach a TmpRas). If you lose much more memory when started from Workbench than from the CLI, make sure your program is not using Exit(n). This would bypass startup code cleanups and prevent a Workbench-loaded program from being unloaded. Use exit(n) instead.

Memory Loss - CLI Only

Some third-party shells dynamically allocate history buffers, or cause other memory fluctuations. Also, if your program executes different code when started from CLI, check that code and its cleanup. And check your startup.asm if you wrote your own.

Memory Loss - Ctrl-C Exit Only

This occurs when you have Amiga-specific resources allocated and you have not disabled your compiler's automatic Ctrl-C handling (causing all of your program clean-ups to be skipped). Disable the compiler's Ctrl-C handling and handle Ctrl-C yourself.

Memory Loss - During Execution

A continuing memory loss during execution can be caused by failure to keep up with all of the IDCMP messages (such as MOUSEMOVE) that you request from Intuition. Intuition can not reuse IDCMP message blocks until you call ReplyMsg() on them. If your window's allotted message blocks are all in use, new ones will be allocated and not freed until the window is closed. Continuing memory losses can also be caused by a program loop containing an allocation/open type call without a corresponding free.

Memory Loss - Workbench Only

This is often caused by the failure of your code to unload after you exit. Make sure that your code is being linked with a correct, standard startup module and do *not* use the Exit(n) function to exit your program. The Exit(n) function will bypass your startup code's cleanup, including its ReplyMsg() of the WorkbenchStartup message (this signals Workbench to unload your program from memory). You should exit via exit(n) where n is a valid DOS error code such as RETURN_OK (*libraries/dos.h*). You may also exit with a final closing brace "}" or with the return statement. Assembler programmers using startup code can JMP to _exit with a long return value on the stack or use the RTS instruction.

Menu Problems

A flickering menu is caused by leaving a pixel or more space between menu subitems in your menu structures. Crashing after browsing a menu (looking at menu without selecting any items) is caused by not properly handling MENUNULL select messages. Multiple selection not working is caused by improper handling of the NextSelect field properly. See the Menus chapter from the *Intuition* manual for more details.

Out-of-Sync Response to Input

This is caused by failing to handle all received signals or messages after waking up from a Wait() or WaitPort() call. More than one event or message may have caused your program to be awakened. Check the signals returned by Wait() and act on every one that is set. At ports which may have more than one message (such as a window's IDCMP port) you must handle the messages in a while(msg=GetMsg(...)) loop.

Performance Loss in Other Processes

This is often caused by a program doing one of the following:

- Busy waiting or polling.
- Running at a higher priority.
- Doing lengthy Forbid()s, Disable()s, or interrupt handling.

Sound Samples Won't Play Correctly

The data for audio samples *must be in chip memory*. Check your compiler manual for directives or flags which will place your audio sample data in chip memory. Also, you can dynamically allocate chip memory and copy or load the audio sample there.

Trackdisk Data Not Transferred

This may occur if your trackdisk buffers are not in chip memory.

Windows - Borders Flicker after Resize

Set the `NOCAREREFRESH` flag. Even `SMART_REFRESH` windows can generate refresh events if there is a sizing gadget, so if you don't have specific code to handle this, you must set the `NOCAREREFRESH` flag. If you do have refresh code, be sure to use the Begin/EndRefresh() calls. Failure to do one or the other will leave Intuition in an intermediate state and slow down operation for all windows on the screen.

GENERAL DEBUGGING TECHNIQUES

Isolate the problem by using printf() to find the section of code in which the problem occurs. If you cannot display messages on the screen, use kprintf() to send messages to the serial port or dprintf() for the parallel port (see Linker Library documentation). Check the initial values, allocation, use, and freeing of all pointers and structures used in the problem area. Also make sure that all of your system and internal function calls pass correct initialized arguments and that all possible error returns are checked for and handled.

Use Debugging Tools

A variety of debugging tools are available to help locate faulty code. There are source level debuggers (such as Lattice's *CodePRobe*), crash interceptors (such as *GOMF*), memory watchdogs like *MemWatch* and *WatchMem*, and other helpful tools like *MemMung*, *Avail*, *WBFrags*, etc.

Test With Different Configurations

Test your program on a wide variety of systems and configurations. Programs with coding errors may appear to work properly on one configuration but may fail or cause fatal problems on another. Make sure that your code is tested on both the 68000 and the 68020/30, on machines with and without fast memory, and on machines with and without enhanced chips. Test all of your program functions on every machine.

Test All Error and Abort Code

A program with missing error checks or unsafe cleanup might work fine when all of the items it opens or allocates are available, but may fail fatally when an error or problem is encountered. Try your code with missing files, filenames with spaces, incorrect filenames, cancelled requesters, Ctrl-C, missing libraries or devices, low memory, missing hardware, etc.

Test all of your text input functions with international ASCII characters (such as the character produced by pressing ALT-F then A). Rawkey codes produce different keyboard characters on the various national keyboards (higher levels of keyboard input are automatically translated to the proper characters). If your program will be distributed internationally, support and take advantage of the additional screen lines available on a PAL system. On A2000s with the enhanced Agnus chip, a PAL display can be selected via motherboard jumper J102. Note that a base PAL machine will have less memory free due to the larger display size.

A

INTERNATIONAL KEYBOARD INPUT

Eric Cotton and Carolyn Scheppner

The Amiga computers are sold internationally with a variety of local keyboards which match the standards of particular countries. The 1.2 Enhancer manual contains diagrams of the keyboards which are currently distributed. If you examine the diagrams, you will notice that some letters and special symbols are in different physical positions on various keyboards. For instance, on the German and Italian keyboards, the "Y" and "Z" keys are swapped when compared to the USA keyboard. Since the physical position of a key determines the raw key code that it generates, straight RAWKEY input is not internationally compatible.

The 1.2 Amiga console device supports national keyboards by providing mapping of raw key codes to the proper acscii characters and strings as specified in a Keymap. In addition, the console device provides handling of the "deadkeys" used to generate accented characters. Any keyboard input which is processed by the console device will be automatically translated to the installed keymap. The new 1.2 Setmap command is used to install the default keymap to be used by the console device. If you wish to experiment with the national keymaps, the KeyToy program on Extras 1.2 provides a graphic simulation of the current various keyboards.

All of the national Keymaps, including USA, contain "deadkeys". A deadkey is generally an ALT/letter combination which specifies that a particular accent mark is to be placed on the next character typed. The ASCII values for these characters are all greater than 127 and they can be seen in the International Character Codes table in the AmigaDOS manual. If a deadkey is used to request an invalid accent for a character, the normal unaccented character results.

If you look at the "usa" keyboard diagram in the Enhancer manual, you'll see that the F, G, H, J, and K keys (shadowed) are defined as deadkeys. In the diagram, various accent marks are printed in the lower half of each of these keys. Pressing ALT and one of these shadowed keys is a request for the illustrated accent to be placed on the next character you type.

For example, with the default usa keymap (try from CLI):

```
ALT/F (') then A  results in an A accented with '
ALT/H (^) then E  results in an E accented with ^
ALT/J (~) then C  results in a plain C (invalid accent for C)
```

Under 1.2, the console.device, IDCMP VANILLAKEY, and AmigaDOS CON: and RAW: all provide automatic handling of deadkeys and translation of raw keycodes to ASCII based on the current keymap. If your software requires non-VANILLA keys such as the cursor and function keys, using console device keyboard input in your Intuition window will allow you to receive the escape sequences generated by these keys. If you are using IDCMP RAWKEY input in international software, you must properly use the console devices's RawKeyConvert() function to get keymap translation and deadkey handling. See the example DeadKeyConvert() in the 1.2 Enhancer manual (P. 65) or the

1.2 Readmes. Use NULL for the kmap argument to get translation to the currently installed Keymap.

Be sure to test your code with ASCII characters > 127 (such as accented characters like ALT/F A) wherever your code accepts keyboard input to insure that your logic and data structures work properly with high ASCII values. In addition, test your code after using SetMap to install various keymaps, using the Enhancer manual keyboard diagrams as a guide for what the keycaps would show in that country. You won't be able to test all of the keys of some national keyboards, but you can test enough keys to be sure you are getting translation.

The following information is a supplement to the Console Device chapter of the Rom Kernel Manual, describing the Keymap format and console.device handling of deadkeys. The page, section, and figure numbers in this supplement are correct for the Addison-Wesley ROM Kernel Manual: Libraries and Devices.

Dead-Class Keys

The term "dead-class key" refers to keys that either modify or can themselves be modified by other dead-class keys. There are two types of dead-class keys: dead and deadable. A dead key is one which can modify the key pressed immediately following. For example, on the German keyboard there is a dead key marked '` that when followed by (for instance) the 'A' key will produce the character 'a' (International Character Code $E0). A deadable key is one that can be prefixed by a dead key. The 'A' key in the previous example is a deadable key. Thus, a dead key can only affect the output of a deadable key.

For any key that is to have a dead-class function, whether dead or deadable, the qualifier KCF_DEAD flag must be included in the entry for the key in the KeyMapTypes table. The KCF_DEAD type may also be used in conjunction with the other qualifiers. Furthermore, the key's KeyMap table entry must contain the longword address of the key's dead-key descriptor data area in place of the usual 4 ASCII character mapping.

Below is an excerpt from the AMIGA 1000 German key map. It will be referenced in the following discussion.

Example 4-2: Dead-Class Keys

```
new LoKeyMapTypes:
        DC.B    KCF_DEAD+KCF_SHIFT+KCF_ALT+KCF_CONTROL  ;aA (Key 20)
                    ...                     ;(more...)
        DC.B    KCF_DEAD+KCF_SHIFT+KCF_ALT+KCF_CONTROL  ;hH (Key 25)
                    ...                     ;(more...)

new LoKeyMap:
        DC.L    key20                       ; a, A, ae, AE
                    ...                     ;(more...)
        DC.L    key25                       ;h, H, dead ^
                    ...                     ;(more...)

;------ possible dead keys
key25:
        DC.B    0,'h',0,'H'                 ;h, H
        DC.B    DPF_DEAD,3,DPF_DEAD,3       ;dead ^
```

```
                DC.B    0,$08,0,$08,0,$88,0,$88 ;control translation
                    ...                         ;(more...)

        ;------ deadable keys (modified by dead keys)
        key20:
                DC.B    DPF_MOD,key20u-key20    ;deadable flag, number of
                                                ;bytes from start of key20
                                                ;descriptor to start of un-
                                                ;shifted data

                DC.B    DPF_MOD,key20s-key20    ;deadable flag, number of
                                                ;bytes from start of key20
                                                ;descriptor to start of shift-
                                                ;ed data

                DC.B    0,$E6,0,$C6             ;null flags followed by rest
                DC.B    0,$01,0,$01,0,$81,0,$81 ;of values (ALT, CTRL...)
        key20u:
                DC.B    'a',$E0,$E1,$E2,$E3,$E4 ;'a' alone and characters to
                                                ;output when key alone is
                                                ;prefixed by a dead key
        key20s:
                DC.B    'A',$C0,$C1,$C2,$C3,$C4 ;SHIFTed 'a' and characters to
                                                ;output when SHIFTed key is
                                                ;prefixed by a dead key
```

In the example, key 25 (the 'H' key) is a dead key and key 20 (the 'A' key) is a deadable key. Both keys use the addresses of their descriptor data areas as entries in the LoKeyMap table. The LoKeyMapTypes table says that there are four qualifiers for both: the requisite KCF_DEAD, as well as KCF_SHIFT, KCF_ALT, and KCF_CONTROL. The number of qualifiers determine length and arrangement of the descriptor data areas for each key. Table 4-8 below shows how to interpret the KeyMapTypes for various combinations of the qualifier bits. For each possible position a pair of bytes is needed. The first byte in each pair tells how to interpret the second byte (more about this below).

Table 4-8: Dead Key Qualifier Bits

If type is:	Then the pair of bytes in this position in the dead-class key descriptor data is output when the key is pressed along with:							
NOQUAL	alone	-	-	-	-	-	-	-
A	alone	A	-	-	-	-	-	-
C	alone	C	-	-	-	-	-	-
S	alone	S	-	-	-	-	-	-
A+C	alone	A	C	A+C	-	-	-	-
A+S	alone	S	A	A+S	-	-	-	-
C+S	alone	S	C	C+S	-	-	-	-
S+A+C (VANILLA)	alone	S	A	S+A	C	C+S	C+A	C+S+A

NOTE: the abbreviations A, C, S stand for ALT, control, and SHIFT, respectively. Also note that the ordering is reversed from that in the normal KeyMap table.

Because keys 20 and 25 each use three qualifier bits (not including KCF_DEAD),

according to the table there must be 8 pairs of data, arranged as shown. Had only KCF_ALT been set, for instance, (not including KCF_DEAD), just two pairs would have been needed.

As mentioned earlier, the first byte of data pair in the a descriptor data area specifies how to interpret the second byte. There are three possible values: 0, DPF_DEAD and DPF_MOD. In Example 4-2 DPF_DEAD appears in the data for key 25, while DPF_MOD is used for key 20. It is the use of these flags which determines whether a dead-class key has dead or deadable function. A value of zero causes the unrestricted output of the following byte.

If the flag byte is DPF_DEAD, then that particular key combination (determined by the placement of the pair of bytes in the data table) is dead and will modify the output of the next key pressed (if deadable). How it modifies is controlled by the second byte of the pair which is used as an index into into part(s) of the data area for ALL the deadable (DPF_MOD set) keys.

Before going further, an understanding of the structure of a descriptor data area wherein DPF_MOD is set for one (or more) of its members is necessary. Referring to the example, we see that DPF_MOD is set for the first and second pairs of bytes. According to its LoKeyMapTypes entry, and using Table 4-8 as a guide, these pairs represent the alone and SHIFTed values for the key. When DPF_MOD is set, the byte immediately following the flag must be the offset from the start of the key's descriptor data area to the start of a table of bytes describing the characters to output when this key combination is preceded by any dead keys. This is where the index mentioned above comes in. The value of the index from a prefixing dead key is used to determine which of the bytes from the deadable keys special table to output. The byte in the index+1 position is sent out. (The very first byte is the value to output if the key was not prefixed by a dead key.) Thus, if ALT'ed H is pressed (dead) and then SHIFTed A, an 'A' will be output. This is because:

- The byte pair for the ALT position of the H key (key 25) is DPF_DEAD,3 so the index is 3.
- The byte pair for the SHIFT position of the A key (key 20) is DPF_MOD,key20s-key20, so we refer to the table-of-bytes at key20s.
- The third+1 byte of the table-of-bytes is $C2, a 'A' character.

Note that the number of bytes in the table-of-bytes for all deadable keys must be equal to the highest index value of all dead keys plus 1.

PAL and Audio

Carolyn Scheppner

To generate accurate frequencies on both NTSC and PAL Amigas, audio programs that calculate periods from frequencies must use the appropriate clock constant on each machine. Programs can determine if they are running on an NTSC or PAL machine by checking GfxBase->DisplayFlags for the PAL or NTSC flags defined in gfxbase.h. The clock constants for period calculations are 3.579545 clocks/second on an NTSC machine and 3.546895 clocks/second on a PAL machine. An example of an integer calculation using these values follows.

```
#define NTSC_CLOCK (3579545)
#define PAL_CLOCK  (3546895)

ULONG clock, clockx100, periodx10, period, hertzx10, sampleBytes;

/* Check GfxBase->DisplayFlags and set clock to appropriate value
 * Set hertzx10 to 10 * desired frequency in hertz
 * Set sampleBytes to number of bytes in the one cycle waveform sample
 */

clockx100 = 100 * clock;
periodx10 = (clockx100 / hertzx10) / sampleBytes;
/* round off */
period = (periodx10 + 5) / 10;
```

Amiga Development Guidelines

The environment of the Amiga computer is quite different than that of many older computers. The Amiga is multitasking, which means multiple programs must share the same machine without interfering with each other. It also means that certain guidelines must be followed during programming.

- Always make sure you actually GET what you ask for. This applies to memory allocations, windows, screens, file handles, libraries, devices, ports, etc. Where an error value or return is possible, ensure that there is a reasonable failure path. Many poorly written programs will *appear* to be reliable, until some error condition (such as memory full or a disk problem) causes the program to continue with an invalid or null pointer, or branch to untested error handling code.

- Always clean up after yourself. This applies for both normal program exit and program termination due to error conditions. Anything that was opened must be closed, anything allocated must be deallocated. It is generally correct to do closes and deallocations in reverse order of the opens and allocations. Be sure to check your development language manual and startup code; some items may be closed or deallocated automatically, especially in abort conditions. If you write in the C language, make sure to provide your own CTRL-C handling to free any Amiga-specific resources and structures.

- Remember that memory, peripheral configurations, and ROMs differ between models and between individual systems. Do not make assumptions about memory address ranges, storage device names, or the locations of system structures or code. Never call ROM routines directly. Beware of any example code you find which calls routines at addresses in the $F00000 range. These are ROM routines and they will move with every OS release. The only supported interface to system ROM code is through the provided library, device, and resource calls.

- Do not assume library bases or structures will exist at any particular memory location. The only absolute in the system is address 0x00000004, which contains a pointer to the exec.library base. Do not modify or depend on the format of private system structures. This include the poking of copper lists, memory lists, and library bases.

- Do not assume that programs can access hardware resources directly. Most hardware is controlled by system software and resources that will not respond well to interference. Shared hardware requires programs to use the proper sharing protocols. Using the defined interface enhances the probability that your software will continue to operate on future Amiga computers.

- Do not access shared data structures directly without the proper mutual exclusion (locking). Remember that other tasks may be accessing the same structures.

- Do not assume that system flags and options are limited to current possible values or choices. For example, do not assume a display must be PAL if not NTSC, and do not assume an event must be SELECTUP if not SELECTDOWN. Explicitly check for the values or choices you support, and provide a default case for everything else (for example a default ReplyMsg() for unknown IntuiMessages).

- The system does not monitor the size of a program's stack. Take care that your program does not cause stack overflow, and provide enough leeway for the possibility that future revisions of system functions might require additional stack space.

- If your program waits for external events like menu selection or key-strokes, do not bog down the multitasking system by busy-waiting in a loop. Instead, let your task go to sleep by Wait()ing on its signal bits. For example:

```
ULONG windowsig  = 1L << window->UserPort->mp_SigBit;
ULONG consolesig = 1L << consoleport->mp_SigBit;
signals = (ULONG) Wait(windowsig | consolesig);
```

  This turns the signal bit number for each port into a mask, then combines them as the argument for the exec.library/Wait() function. When your task is awakened, handle all of the messages at each port where the SigBit is set. There may be more than one message per port, or no messages at the port. Make sure that you ReplyMsg() to all messages that are not replies themselves.

- Tasks (and Processes) execute in 68000 processor user mode. Supervisor mode is reserved for interrupts, traps, and task dispatching. Take extreme care if your code executes in supervisor mode. Exceptions while in supervisor mode are deadly.

- Most system functions require a particular execution environment. All DOS functions and any functions that might call DOS (such as the opening of a disk-resident library, font, or device) can only be executed from a process. A task is not sufficient. Most other ROM Kernel functions may be executed from tasks. Only a few may be executed from interrupts.

- Do not disable interrupts or multitasking for long periods. If you use Forbid() or Disable(), be aware that use of any system function that Waits will temporarily suspend your Forbidden or Disabled state, and allow multitasking and interrupts to occur. Such functions include almost all forms of DOS and device IO, including common "stdio" functions like "printf".

- Do not tie up system resources unless it is absolutely necessary. For example, if your program does not require constant use of the printer, open the printer.device only when you need it. This will allow other tasks to use the printer while your program is running. You must provide a reasonable error response if a resource is not available when you need it.

- Check for memory loss. Operate your program, then exit. Write down the amount of free memory. Repeat the operation of your program and exit. The amount of free memory remaining should be *exactly* the same. Any difference may signal some serious problem in your cleanup. A useful tool for memory testing is the "LoadWB -debug" command; this will start the Workbench tool with a special invisible debug menu. The "flushlibs" option of this menu can cause unused libraries and devices to be flushed out of memory. (The "debug" option invokes the ROM debugger, RomWack, on the serial port at 9600 baud.)

- All data for the custom chips *must* reside in CHIP type memory. This includes bitplanes, sound samples, trackdisk buffers, and images for sprites, bobs, pointers, and gadgets. (On the current generation of machines, CHIP memory is the lowest 512K or 1 Meg of memory in the system.) The AllocMem() call takes a flag (MEMF_CHIP) for specifying CHIP type memory. On machines with expansion (FAST) memory, programs will by default load into FAST memory, and allocations which are not specified as MEMF_CHIP will receive FAST memory. On machines with CHIP memory only, all program code and data, and all allocations will automatically be in CHIP ram, and this can mask the symptoms of improper placement of custom chip data or buffers. Most compilers have options to mark specific data structures or object modules so that they will load into CHIP ram. Some older compilers provide the Atom utility for marking object modules. If this method is unacceptable, use the AllocMem() call to dynamically allocate CHIP memory, and copy your data there. When making allocations that do not require CHIP memory, do not explicitly ask for MEMF_FAST. Ask for memory type 0L or MEMF_PUBLIC as appropriate. If FAST memory is available, you will get FAST memory.

- Do not use software delay loops! Under the multitasking operating system, the time spent in a loop can be better used by other tasks. Even ignoring the effect of multitasking, timing loops are inaccurate and will wait widely varying amounts of time depending on the configuration and processor of the computer. The timer.device can provide accurate timing for use under the multitasking system. The AmigaDOS Delay() function provides a simple interface for longer delays. The 8520 I/O chips provide timers for developers who are bypassing the operating system (see the *Amiga Hardware Reference Manual* for more information).

- Obey structure conventions!

  - All non-byte fields must be word aligned.

  - All address pointers should be 32 bits (not 24 bits). The upper byte must never be used for data.

  - Fields that are not defined to contain particular initial values *must* be initialized to zero. This includes pointer fields.

  - All reserved or unused fields *must* be initialized to zero for future compatibility.

  - Data structures to be accessed by the custom chips, public data structures (such as a task control block), and structures which must be longword aligned must NOT be allocated on a program's stack.

  - Dynamic allocation of structures with AllocMem provides longword aligned memory of a specified type with optional initialization to zero, which is useful in the allocation of structures.

For 68010/68020/68030 compatibility

Special care must be taken to be compatible with the entire family of 68000 processors:

- Do not use the upper 8 bits of a pointer for storing unrelated information. The 68020 uses all 32 bits for addressing.

- Do not use signed variables or signed math for addresses.

- Do not use self-modifying code.

- Do not use software delay loops, and do not make assumptions about the order in which asynchronous tasks will finish.

- The stack frame used for exceptions is different on each member of the 68000 family. The type identification in the frame must be checked! In addition, the interrupt autovectors may reside in a different location on processors with a VBR register.

- Do not use the "MOVE SR,..." instruction! This 68000 instruction acts differently on other members of the 68000 family. If you wish a copy of the processor condition codes, use the exec.library/GetCC() function.

- Do not use the CLR instruction on a hardware register which is triggered by access. The 68020 CLR instruction does a single Write access. The 68000 CLR instruction does a Read access first, then a Write access. This can cause a hardware register to be triggered twice. Use MOVE(.whatever) #0, address instead.

Hardware Programming Guidelines

If you find it necessary to program the hardware directly, then it is your responsibility to write code which will work properly on various various models and configurations. Be sure to properly request and gain control of the hardware resources you are manipulating, and be especially careful in the following areas:

- All custom chip registers are READ ONLY or WRITE ONLY. Do not read Write-only registers, and do not write to Read-only registers.

- Do not write spurious data to, or interpret undefined data from, currently unused bits or addresses in the custom chip space. To be software-compatible with future chip revisions, all undefined bits must be set to zeros on writes, and must be masked out on reads before interpreting the contents of the register.

- Do not write past the current end of custom chip space. Custom chips may be extended or enhanced to provide additional registers, or to use bits which are currently undefined in existing registers.

- Do not read, write, or use any currently undefined address ranges. The current and future usage of such areas is reserved by Commodore and is subject to change.

Additional Assembler Development Guidelines

- Do not use the "TAS" instruction on the Amiga. System DMA can conflict with this instruction's special indivisible read-modify-write cycle .

- System functions must be called with A6 containing the library or device base. Libraries and devices assume A6 is valid at the time of any function call. Even if a particular function does not currently require its base register, you must provide it for compatibility with future system software releases.

- Except as noted, system library functions use registers D0, D1, A0, and A1 as scratch registers and you must consider their former contents to be lost after a system library call. The contents of all other registers will be preserved. System functions which provide a result will return the result in D0.

- System functions that return a result may not necessarily affect the processor condition codes. The caller must test the returned value before acting on a condition code. This is usually done with a TST or MOVE instruction.

Some Caveats to Hardware Level Programmers

The Amiga is available in a variety of models and configurations, and is further diversified by a wealth of add-on expansion peripherals and processor replacements. In addition, even standard Amiga hardware such as the keyboard and floppy disks, are supplied by a number of different manufacturers and may vary subtly in both their timing and in their ability to perform outside of their specified capabilities.

The Amiga operating system is designed to operate the Amiga hardware within spec, adapt to different hardware and RAM configurations, and generally provide upward compatibility with any future hardware upgrades or ``add ons'' envisioned by the designers.
For maximum upward compatibility, it is strongly suggested that programmers deal with the hardware through the commands and functions provided by the Amiga operating system.

If you find it necessary to program the hardware directly, then it is your responsibility to write code which will work properly on various models and configurations. Be sure to properly request and gain control of the hardware you are manipulating, and be especially careful in the following areas:

Do not jump into ROM. Beware of any example code that calls routines in the $F80000 to $FFFFFF range. These are ROM addresses and the ROM routines WILL move with every OS revision. The only supported interface to system ROM code is through the provided library, device, and resource calls.

Do not modify or depend on the format of any private system structures.

This includes the poking of copper lists, memory lists, and library bases.

Do not depend on any address containing any particular system structure or type of memory. The system modules dynamically allocate their memory space when they are initialized. The addresses of system structures and buffers differ with every OS, every model, and every configuration, as does the amount of free memory and system stack usage. Remember that all data for direct custom chip access must be in CHIP RAM. This includes bit images (bitplanes, sprites, etc), sound samples, trackdisk buffers, and copper lists.

Do not write spurious data to, or interpret undefined data from currently unused bits or addresses in the custom chip space. All undefined bits must be set to zero for writes, and ignored on reads.

Do not write data past the current end of custom chip space.

Custom chips may be extended or enhanced to provide additional registers, or to use currently undefined bits in existing registers.

All custom chip registers are read only OR write only. Do not read write only registers, and do not write to read only registers.

Do not read, write, or use any currently undefined address ranges. The current and future usage of such areas is reserved by Commodore and is definitely subject to change.

If you are using the system libraries, devices, and resources, you must follow the defined interface. Assembler programmers (and compiler writers) must enter functions through the library base jump tables, with arguments passed as longs and library base address in A6. Results returned in D0 must be tested, and the contents of D0-D1/A0-A1 must be assumed gone after a system call.

> The assembler TAS instruction should not be used in any Amiga program. The TAS instruction assumes an indivisible read-modify-write but this can be defeated by system DMA. Instead use BSET and BCLR. These instructions perform a test and set operation which cannot be interrupted.
>
> TAS is only needed for a multiple CPU system. On a single CPU system, the BSET and BCLR instructions are identical to TAS, as the 68000 does not interrupt instructions in the middle. BSET and BCLR first test, then set bits.

Do not use assembler instructions which are privileged on any 68000 family processor, most notably MOVE SR,<ea> which is privileged on the 68010/20/30. Use the Exec function GetCC() instead of MOVE SR, or use the appropriate non-privileged instruction as shown below:

CPU	Usr Mode	Super Mode
68000	MOVE SR,<ea>	MOVE SR<<ea>
68010/20/30	MOVE CCR,<ea>	MOVE SR<<ea>

All addresses must be 32 bits. Do not use the upper 8 bits for other data, and do not use signed variables or signed

math for addresses. Do not execute code on your stack or use self-modifying code since such code can be defeated by the caching capabilities of some 68xxx processors. And never use processor or clock speed dependent software loops for timing delays. See Appendix F of the Amiga Hardware Reference Manual for information on using an 8520 timer for delays.

When strobing any register which responds to either a read or a write, (for example copjmp2) be sure to use a MOVE.W #$00, not CLR.W. The CLR instruction causes a read and a clear (two accesses) on a 68000, but only a single access on 68020 and above. This will give different results on different processors.

If you are programming at the hardware level, you must follow hardware interfacing specifications. All hardware is NOT the same. Do not assume that low level hacks for speed or copy protection will work on all drives, or all keyboards, or all systems, or future systems. Test your software on many different systems, with different processors, OS, hardware, and RAM configurations.

CDTV CD-ROM File Structure

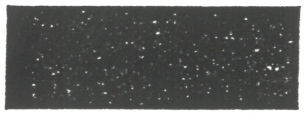
CDTV Developer Notes Issue #1
6/8/90

CD-ROM File Structure

The following section is a "Work-in-Progress" document.

This section covers the actual CDTV CD-ROM file system, CD Device Driver Include File, and an article on the 512K Byte Logical Block Size. Like other parts of this first installment of Developer Notes this section is far from complete. We will be adding more information on the CD-ROM file structure in future issues.

We would like your feedback and comments on this material. If you have questions, comments, or suggestions then contact Gail Wellington through USENET... rutgers/uunet!cbmvax!gail, BIX... gwelly, or FAXing your ideas... 215-431-9156 (no phone calls please).



COMMODORE AMIGA CD ROM FILE SYSTEM

Definition, Specifications, Developer Guidelines

Preliminary Draft, March 26, 1990

1. Project Definition 1
   - 1.1 Scope of System 1
   - 1.2 Target Machines 1
   - 1.3 Implementation Modules 2
   - 1.4 Other Goals 2
   - 1.5 Schedule & Dependencies 2

2. Disk Format 3
   - 2.1 ISO Summary 3
   - 2.2 Amiga Specifics 4

3. Program Interface 6

4. Bootstrap 6
   - 4.1 ROM Bootstrap 7
   - 4.2 Disk Bootstrap 7
   - 4.3 Redundancy 7
   - 4.4 Availability of Code 8

5. CDFS Options 8
   - 5.1 Memory Usage 8
   - 5.2 Performance 8
   - 5.3 Redundancy 8
   - 5.4 Error Handling 9

6. Disk Geography 9

©Commodore-Amiga, Inc., 1990

By Carl Sassenrath, Consultant
Sassenrath Research, Ukiah, CA
Phone: 707-462-4878 (FAX: 4879)

Confidential Document – Not for Distribution

AMIGA CD ROM FILE SYSTEM

Project Definition, Specifications, and Developer Guidelines

1. Project Definition

The primary goal of this project is to develop a CD based Amiga compatible File System for general use with new hardware being developed by Commodore. For the purposes of documentation and discussion we will adopt the abbreviation **CDFS** for this system.

1.1 Scope of System

CDFS is not a complete file system – it is not a replacement for the higher-level AmigaDOS operating system. It operates as a standard, mountable AmigaDOS device handler, and thus preserves the fundamental run-time environment for the broad base of application software.

The CDFS project will define, develop, and provide:

- A DOS device handler as an extended module of code with an AmigaDOS message based interface.
- A standard format for all compact disks mounted as volumes under CDFS.
- A bootstrap mechanism for machines with a compact disk as the primary system volume.
- Guidelines for optimizing performance and controlling file system behavior for specific types of applications.

1.2 Target Machines

CDFS is designed primarily for use on the Amiga *Baby*, where it will be the resident file system for the majority of applications. It will be capable of bootstrapping and servicing the relevant portions of AmigaDOS, and it may operate alone as the sole system storage handler or in conjunction with other storage handlers.

For native Amiga machines with external CD devices, the CDFS will provide its services as a foreign file system for access to CD volumes. In such configurations CDFS volumes will not operate as bootstrap system devices and may or may not be fully operational in all cases (e.g. CD XA).

1.3 Implementation Modules

Two software modules will be developed to support the operation of CDFS on the *Baby*:

1. Within the machine's ROM, a small bootstrap file system module will be needed to verify the disk format, set system options, and load the disk resident bootstrap module. This code only accesses the disk as a set of logical sectors – it does not access files[1].

2. Every bootstrap disk will include a fully functional CDFS[2] that gets loaded into memory during the bootstrap process. This code will allow the disk to be accessed as a set of files and directories resembling an AmigaDOS volume.

1.4 Other Goals

In addition to the development of the items mentioned above, the CD File System is intended to:

- Provide outstanding performance for applications tuned to the guidelines outlined in this specification (including the Amiga Workbench).
- Allow application developers flexibility with regard to system configuration, operations, and memory usage.
- Be completely reliable – as expected of a mass produced and widely distributed operating system.

1.5 Schedule & Dependencies

We don't have much time. This specification needs to be finalized within a few weeks. The critical date for scheduling purposes is the C.E.S. demo the first week in June 1990. Working backwards the CDFS must be operational in an Alpha Test form by the end of April.

Our primary dependency in reaching this goal will be the availability of test hardware. To reduce the critical path we have developed a "CD RAM" device driver which looks like a CD in its physical format.

A second, but equally important, dependency is the CD disk device driver. Any delay in receiving a functioning driver will delay the project (but should not compound the delays).

---

[1] It may also provide essential services to other ROM based applications as necessary.

[2] This is also where you might load a proprietary operating system if you don't want to operate under standard AmigaDOS.

A final dependency not to be overlooked is the origination software. This software must be working to some degree for CDFS disks to be created.

2. Disk Format

The standard AmigaDOS disk structures found on most magnetic media do not offer sufficient performance for CD ROMs. This is primarily due to the high overhead of seeking on the current technology CD hardware. As a result, we have determined that the Amiga would benefit most by adopting the International Standard for CD ROMS, ISO 9660.

2.1 ISO Summary

This section briefly summarizes the ISO 9660 standard for those unfamiliar with its details.

ISO 9660 specifies a disk volume structure and a hierarchical file directory structure for use on all types of CD ROMS. Among other things it specifies the attributes of volumes, the possible relationships among volumes, the placement of files, and the attributes of files.

Volume Organization

The standard defines the logical addressing of the volume to be *logical sectors* of 2048 bytes, each of which can be subdivided down to *logical blocks* of 512, 1024, or 2048 bytes. For the most part, locations on the disk are referred to by *logical block number* (LBN).

All volumes are divided into a *system area* (the first 16 sectors of a disk) and a *data area* (the rest of the disk). The use for the system area is not defined. The data area is broken down into volume descriptors, path tables, directory descriptors, file descriptors, and file data itself.

Volume Descriptors

Volume descriptors are used for storing information describing a particular volume or set of volumes. There are several types of volume descriptors, of which there may be any number. For example a volume may contain many different descriptors, with each descriptor being specific to a particular delivery system. Systems may ignore descriptors not meant for them.

The *Primary Volume Descriptor* includes the volume name, target system name, volume size, publisher name, copyright data, relevant dates, pointers to various volume structures, the root directory entry, and other items.

Another descriptor worth mentioning is the *Boot Record*. This structure contains a target system name and the bootstrap code needed for that system.

Finally, *Volume Partitions* can be used for dividing a disk into CD Audio, CD XA, CD+G, and other non-file areas.

Directory Structure

Like most file system formats, the directory structure includes information to identify a file, describe its attributes, and locate it on disk. Directories can form a hierarchy with a maximum nesting depth of eight.

In addition to the directories, a special structure is provided for fast access to a particular directory. The *Path Table* provides the system with a means of accessing a particular directory with a single disk seek.

File Structure

Files have a flexible structure within the standard. A file may be broken into multiple extents, spread over multiple volumes, or even interleaved, but the data content of a file is not regulated by the standard.

A file may contain an *Extended Attribute Record*, XAR, which specifies additional information about the file: owner, group, permissions, record format, record attributes, system identifier, creation date, modification date, expiration date, effective date, record format information, and several other types of system structures.

2.2 Amiga Specifics

The capabilities of AmigaDOS are a subset of the total flexibility allowed by ISO 9660. This enables us to create an ISO volume that functions properly with AmigaDOS, even though the volume contains no AmigaDOS structures of any kind. It will be the task of CDFS to translate and synthesize the formats and interfaces needed by AmigaDOS.

With regard to Amiga CDFS V1.0, the ISO standard format is interpreted and applied as follows.

General

ISO Interchange Level 2 (only one File Section per file), and ISO Implementation Level 1 (ignore supplementary volume descriptors).

The logical block size will be 512 bytes. (This has no impact on performance, because device accesses will be in full 2K sectors.)

System Area

Developers should not expect to use the System Area (sectors 0 to 15) for Amiga code or data when the CD is being originated for multiple delivery systems.

Volume Descriptors

At least one Primary Volume Descriptor is required. This descriptor shall include the system identifier string:

```
"COMMODORE_AMIGA"
```

The Primary Volume Descriptor should include valid data for all ISO fields including:

```
Volume Identifier
Copyright File Identifier
Creation & Modification Dates
```

If a volume is bootable, it will contain at least one Boot Record with the above system identifier. The boot record will contain additional information and will be defined later in this document.

Volume Partition Descriptors will use the same system identifier[3] and will specify a Volume Partition Identifier.

Path Table

Path Tables will conform to the ISO standard with one possible exception regarding foreign character sets (e.g. German, French, etc). This needs further investigation to determine the full impact of the ISO standard on foreign character sets.

Directories

Here again we doubt the wisdom of the ISO standard: it does not easily accommodate file names with lower case or alternate characters. We may vary from the standard in this regard if the alternatives are overly complex (but only at the option of the publisher.) This is still under investigation.

Even though AmigaDOS accommodates a directory hierarchy of more than eight levels, CDFS volumes will be restricted to eight in order to comply with ISO 9660.

In the File Flags (BP 26), with the exception of the Directory flag all other flags will be ignored (at least for the first version of

---

[3]I'm still uncertain exactly how we are going to implement partitions.

CDFS). This means that CDFS 1.0 will not support associated files, special record formats, file protection/ownership, or multiple extents.

The *Recording Date and Time* can be used to store the AmigaDOS file creation date (in the ISO format however).

Files

Files will contain only file data. There will be no AmigaDOS structures embedded within the body of a file. This will permit large reads of data directly into application buffers.

CDFS 1.0 will not support files broken into multiple extents or spread over multiple volumes. Supporting file interleaving is still under investigation.

CDFS 1.0 will allow the existence of *Extended Attribute Records* in files, but will skip and ignore their contents.

AmigaDOS File Comments will be placed in the *System Use* field of a directory entry; however, there may be some restrictions (being investigated). The presence of comments will have a negative effect on performance for moderate to large directories.

3. Program Interface

Because it is an AmigaDOS File Handler, the CDFS does not present a new interface to application programmers. For the most part, it remains hidden under the DOS file functions: Open, Close, Read, Write, Seek, etc.

Obviously, CD ROM volumes impose restrictions when it comes to write or modify types of functions. For example, files may not be renamed or deleted, nor may they have their dates, comments, or permissions changed.

CDFS includes a few configuration options as part of the Primary Volume Descriptor, that control the effects of errors produced when from writing or modifying a CD volume. Rather than generate an error message or requestor, options may be selected to ignore the operation[4]. These options should be used with care.

4. Bootstrap

On the *Baby*, CDFS will normally require a two stage bootstrap. The first stage involves reading the volume descriptors, looking

---

[4]It is conceivable that a publisher may want to port a program without eliminating all of the code that writes to disk. Try using a Macintosh CD Hypercard for a couple hours as a justification for this.

for a Bootstrap Record, and loading it into memory. The second stage involves reading files and bring up the full system.

4.1 ROM Bootstrap

The first stage bootstrap resides in ROM. Its purpose is to determine the format of a CD ROM, set file system options, and load the disk resident bootstrap module. This ROM bootstrap only accesses the disk as a set of logical sectors – it does not know how to access files.

It begins by scanning sectors looking for a volume descriptor that contains the Amiga system identifier. Finding a valid Primary Volume Descriptor (PVD) certifies that the volume can be read by the Amiga. Finding a valid Bootstrap Record allows the system to be bootstrapped further.

Upon finding a valid PVD, the ROM bootstrap loads a CDFS system attribute structure from the *Application Use* portion of the PVD. This structure contains file system options that control block caching, directory sizing, error handling, etc. It can be empty, in which case the ROM defaults are used.

4.2 Disk Bootstrap

When a valid Bootstrap Record is located, its *Boot Identifier* is examined. This string indicates for which version of the Amiga this boot record is intended. The first version of *Baby* will contain the string:

```
"AMIGA_CD_01"
```

If a valid identifier is found, the code following it (BP 72) is loaded into memory and executed.

For CDFS, this code will read the CDFS handler off disk into memory and link it to DOS. This code can be located anywhere on disk and is not necessarily accessible as a normal file on the volume.

Of course a publisher may provide his own bootstrap code which will load special applications without the need for CDFS.

4.3 Redundancy

Unless a publisher wishes to have a number of unhappy customers, it is advised that multiple copies of the required volume descriptors be provided on disk, along with multiple copies of the disk bootstrap, path tables, etc.

The ROM bootstrap will ignore volume descriptors that are read with errors. It will act only on the first valid, error free descriptors.

4.4 Availability of Code

The disk bootstrap code (binary) will be provided to developers for inclusion on their disks by either the origination system vendors or directly from Commodore[5].

5. CDFS Options

The behavior of CDFS can be strictly controlled through a special set of options embedded within the Primary Volume Descriptor. During disk origination, these options can be installed to best suit your type of application or run-time environment.

5.1 Memory Usage

A CD ROM developer has nearly complete control over the way CDFS utilizes the Amiga's main memory. Its behavior can span from using very large areas for optimized access to using virtually no memory in specialized applications.

During the ROM bootstrap process, CDFS will utilize approximately 64K of memory as a read buffer. This large space is necessary to insure (for practical cases) that the volume descriptors and bootstrap code are captured in a single CD access. If the system continues to boot without encountering contrary memory options, this buffer will be reused by CDFS for its block buffers.

Once running, CDFS will allocate additional memory for internal structures, AmigaDOS structures, the Path Table, and a directory block pool. CDFS options can be used to regulate the sizes of these areas.

5.2 Performance

To increase file system performance, CDFS often posts reads for more data than has been requested of it. In such cases it tries to anticipate what will be needed next. This option has tremendous value to developers – they control the physical layout of each disk – they determine what gets read in advance. The size of these anticipatory reads can be selected with CDFS options.

Also, depending on the hardware and device driver, options may exist for other types of read options (e.g. target sector reads, or direct application buffer transfers).

5.3 Redundancy

Although the CD hardware will provide a certain degree of error detection and correction, it is advisable that the developer consider adding redundancy where possible. CDFS will incorporate a few mechanisms of recovery for bad accesses[6]. These can be selected with CDFS options.

---

[5]It is our hope that this code be provided free of charge.

[6]Including entirely redundant file system partitions.

5.4 Error Handling

As mentioned earlier, a developer can determine what happens when an application attempts to write or modify the disk. CDFS options will allow separate control over error responses for volume label changes, directory changes, file modifications, etc.

Note that these options apply to the volume overall. It may be desirable to provide these for specific directories or files, and we are investigating the possibilities.

6. Disk Geography

As with any CD ROM, the organization of data on the disk will have a great impact on the performance of the system. During the origination process, a developer can place files in positions that will optimize their access during program execution.

With CDFS a careful and well thought out disk layout can have tremendous performance advantages. A few of suggestions to consider are:

- Group related files together. If you know the order in which your application reads certain files, group them accordingly, and let the file system read several files at once with anticipatory reads.
- Grouping directories with their files. The system normally reads a directory before it finds a file. If the file happens to be read into system buffers during the directory read, another access may not be needed.
- Group together small files. The idea is to get as many files into the buffers as possible within a single read.

As a profitable application of these suggestions, consider an Amiga Workbench volume. Most files on such a disk would be accompanied by an associated ".info" file. You have no doubt experienced the frustration of opening drawers and watching icons appear.

If the above rules were applied, the .info files would be grouped immediately after the directory. Upon reading the directory with the proper CDFS options set, many of the .info files would be read into memory at the same time. The drawer would open and icons would appear almost instantaneous.

```
/*
** CD Device Driver Include File
**
**      Copyright 1990 Commodore-Amiga, Inc
**      Currently Confidential
**      Entirely subject to change.
**
**      Driver Version X.9, 16 May 90
**      By Sassenrath Research, Ukiah, CA
**      Phone: 707-462-4878 FAX: 462-4879
*/


/*
** CD Device Name
**
**      NOTE: Temporary name used for CES to prevent locking in
**      non-finalized interface specifications.  This name will
**      change when the device is finished.
*/
#define CD_NAME "cescd.device"


/*
** CD Device Driver Commands
*/
#define CD_RESET                1
#define CD_READ                 2
#define CD_WRITE                3
#define CD_UPDATE               4
#define CD_CLEAR                5
#define CD_STOP                 6
#define CD_START                7
#define CD_FLUSH                8
#define CD_MOTOR                9
#define CD_SEEK                10
#define CD_FORMAT              11
#define CD_REMOVE              12
#define CD_CHANGENUM           13
#define CD_CHANGESTATE         14
#define CD_PROTSTATUS          15
#define CD_GETNUMSECTORS       16
#define CD_GETSECTORSIZE       17
#define CD_GETDISKSIZE         18
#define CD_EJECT               19
#define CD_ADDCHANGEINT        20
#define CD_REMCHANGEINT        21
#define CD_DRIVECMD            22
#define CD_DRIVESTAT           23
#define CD_TOCMSF              24
#define CD_TOCBLOCK            25
#define CD_PLAYMSF             26
#define CD_PLAYBLOCK           27
#define CD_PLAYTRKIDX          28
#define CD_STOPAUDIO           29
#define CD_PAUSEAUDIO          30
#define CD_RESUMEAUDIO         31
#define CD_SUBQMSF             32
#define CD_SUBQBLOCK           33
#define CD_AUDIOSTAT           34
```

```
/*
** CD Table of Contents
**
**      The CD_TOCMSF and CD_TOCBLOCK comands return an array
**      of this structure.
**
**      Notes:
**
**       1) The first entry (zero) contains special disk info:
**          the Track field indicates the first track number; the
**          LastTrack field contains the last track number, which is
**          only valid for this first entry; and the Position field
**          indicates the last position (lead out area start) on
**          the disk.
**
**       2) The last entry is followed with an entry containing a NULL
**          Track number as a terminator.
*/
struct CDTOC
{
        UBYTE   rsvd;           /* not used     */
        UBYTE   AddrCtrl;       /* SubQ info    */
        UBYTE   Track;          /* Track number */
        UBYTE   LastTrack;      /* See Note!    */
        ULONG   Position;       /* MSF or BLOCK */
};


/*
** CD Digital Audio SubQ
*/
struct CDSubQ
{
        UBYTE   Status;         /* Audio status */
        UBYTE   AddrCtrl;       /* SubQ info    */
        UBYTE   Track;          /* Track number */
        UBYTE   Index;          /* Index number */
        ULONG   DiskPosition;   /* Position from start of disk  */
        ULONG   TrackPosition;  /* Position from start of track */
        UBYTE   ValidUPC;       /* Flag for product identifer   */
        UBYTE   pad[3];
};
```

RATIONALE FOR 512 BYTE LOGICAL BLOCK SIZE

Carl Sassenrath, Consultant for Commodore-Amiga, Inc.
April 26, 1990
Confidential Document & Trade Secrets

Although feedback on the Amiga CD-ROM Filing System (CDFS) Specification has been minimal, some developers have questioned the use of a 512 byte logical block size (LBS). I think it might be helpful at this time for everyone to understand the rationale behind this decision.

The ISO 9660 CD-ROM standard allows logical block sizes of 512, 1024, and 2048 bytes. Although a block size of 2048 is the most common, any of these may be used, so all must be supported by ISO delivery systems, Amiga included.

AmigaDOS uses a file system based on a 512 byte LBS. If you mount a device with larger physical blocks, AmigaDOS (prior to 2.0) chops them up into logical blocks of 512.

For the CDFS, I decided to stay with the AmigaDOS 512 size primarily for performance reasons, but also because it is well tested on the Amiga. Generally it provides

Better main memory cache utilization

Increased chance of finding a block in the cache (better hit rate)

Improved overall disk transfer rate

A size that has been in use and tested on the Amiga for many years

Each of these will be explained in more detail below.

1. Better Main Memory Utilization

The Amiga CDFS buffers its data blocks in a main memory cache. This cache is divided into a number of equal size bins, each of which may contain an arbitrary disk block. Because of the equal size bins, such caches have no problems with external fragmentation, but they do exhibit what is known as internal fragmentation. Simply enough, this is the unused "padding" space required to align the last block in a file with the next block boundary.

It follows that the smaller the size of a block, the less padding would be potentially required to align it with the next block. For example, a one byte file would need 511 bytes of padding for a 512 LBS, and 2047 bytes of padding for a 2048 LBS. You can conclude that the smaller the LBS, the less main memory is wasted from internal fragmentation.

As a more realistic example, the table below compares the number of memory bytes wasted for a few random files actually used in AmigaDOS 1.3. Notice the difference in bytes wasted for a 512 LBS and a 2048 LBS.

FILE NAME	FILE SIZE	WASTE 512	WASTE 2048
Format	12840	192	552
Format.info	290	222	1758
Notepad	50832	144	1680
Notepad.info	302	200	1846
Echo	992	32	1056
Copy	9848	120	1656

Given the large number of ".info" files found on most Amiga disks, this kind of waste adds up. Using the 2048 LBS, opening the Workbench **Utilities** drawer would waste more than 16K bytes of main memory due to the internal fragmentation of the ".info" files alone. With main memory in short supply, this degree of waste is not acceptable.

Some Amiga CD developers point out that with a 512 LBS, the beginning of files will often not align with a physical block on disk, which results in an extra sector being read from disk. With the Amiga CDFS this is not a problem. For efficiency reasons, the CDFS must read as much data off the disk as possible, so it reads and buffers multiple sectors anyway. This is due to the high cost of seeking relative to the cost of data transfer. Once we have located the file on disk, whether we read one sector or two is insignificant compared to the seek time. In addition, a developer could eliminate this problem entirely with a good pre-mastering program (by automatically aligning blocks when necessary).

In addition to saving main memory space, CD-ROM volume space is also conserved with the smaller block size--although it may be considered insignificant when compared to the total size of a volume.

2. Better Cache Hit Rate

A simple fact: Four times more 512 byte blocks fit into main memory than 2048 byte blocks. Having a smaller LBS increases the granularity of buffering, which lets us pack more files into memory. Given the way CDFS reads files, this results in a better chance of a file block being found in memory, thus reducing the need to go to the disk.

For example, when a directory is opened, the CDFS performs an anticipatory read of the blocks following the directory. For a file drawer, this extra read will bring in a large number of the .info files and will cause the drawer's icons to appear within a

split second. This same technique used with a 2048 LBS would require as much as four times the memory for the same net effect.

3. Improved Transfer Rate

With a 512 byte LBS we can pack more files into the same number of physical sectors. This gives us a greater transfer rate because fewer block alignment bytes are read in the process. With the CDFS directory scheme and a 512 LBS, the data transfer rate can, at times, be at four times greater than that achieved with a 2048 LBS.

4. A Well-Tested Size

AmigaDOS release 1.3 (and earlier systems) uses blocks of 512 bytes. Although it may not be optimal in all cases, we know it works. To support a larger block size at the AmigaDOS system level would require release 2.0, and this is not as well tested.

5. A Final Note

As it turns out, the 512 byte LBS is only the recommended size. ISO disks are designed in such a way that the CDFS can read one with a 2048 LBS without needing AmigaDOS Release 2.0. The File Handler will convert AmigaDOS Keys (block pointers) into logical block numbers as part of its job for ISO 9660 compatibility.

A developer who wants to use a 2048 LBS is free to do so, but it would be wise to consider the performance consequences. If the disk contains many small files, a 512 LBS could provide substantially improved performance for most applications.

```
/*************************************************************************
**
**     SIMPLE EXAMPLE CD-DA PROGRAM
**
**     May 1990, Sassenrath Research, Ukiah, CA USA
**
*************************************************************************/

#include <exec/types.h>
#include <exec/io.h>
#include "/cd.h"

extern  void    *malloc();
extern  struct IOStdReq *CreateStdIO();
extern  struct MsgPort  *CreatePort();
extern  BOOL QuitFlag;

BOOL    DevOpen = FALSE;
struct  IOStdReq *IOReq = NULL;
struct  MsgPort  *IOPort = NULL;

main()
{
        InitCD();
        PlayMSF(0x030A00,0x041E00); /* Play 3:10:00 to 4:30:00 */
        PlayMSF2(0x030A00,0x041E00); /* Play 3:10:00 to 4:30:00 */
        QuitCD(0);
}

InitCD()
{
        IOPort = CreatePort(0,0);
        if (IOPort == NULL) QuitCD("Cannot create message port");
        IOReq  = CreateStdIO(IOPort);
        if (IOReq == NULL) QuitCD("Cannot create IO request");
        if (OpenDevice(CD_NAME,0,IOReq,0) != NULL)
                QuitCD("Cannot Open CES CD Device");
        DevOpen = TRUE;
}

QuitCD(s)
        char *s;
{
        if (s) printf("\nERROR: %s\n", s);
        if (DevOpen) CloseDevice(IOReq);
        if (IOReq)  DeleteStdIO(IOReq);
        if (IOPort) DeletePort(IOPort);
        if (s) exit(100); else exit(0);
}

/*
** Synchronous style play.  Command returns when done.
*/
PlayMSF(StartMSF,EndMSF)
        long StartMSF,EndMSF;
{
        IOReq->io_Command = CD_PLAYMSF;
        IOReq->io_Offset  = StartMSF;
        IOReq->io_Length  = EndMSF;
        IOReq->io_Data    = 0;
        IOReq->io_Flags   = 0;
        if (DoIO(IOReq) != 0) QuitCD("Cannot play audio");
```

```
        puts("Audio done.");
}

/*
** Asynchronous style play. Command returns immediately.
*/
PlayMSF2(StartMSF,EndMSF)
        long StartMSF,EndMSF;
{
        int error;

        IOReq->io_Command = CD_PLAYMSF;
        IOReq->io_Offset  = StartMSF;
        IOReq->io_Length  = EndMSF;
        IOReq->io_Data = 0;
        IOReq->io_Flags   = 0;
        SendIO(IOReq);  /* start playing and return immediately */

        /*
        ** Go off and do other things while playing....
        ** If necessary, check back to see how play is doing
        ** with the CheckIO() function.
        */
        while (TRUE)        /* do something */
        {
                /* Check for normal termination of audio.
                ** Only needed if you have got other things to
                ** do and just want to check.  You should not
                ** loop on this condition (use WaitIO()).
                */
                if (CheckIO(IOReq) != NULL)
                {
                        puts("Audio done.");
                        break;
                }

                /* Check for an early termination of audio: */
                if (QuitFlag)  /* ....set by ctrl-C or whatever */
                {
                        puts("Cancel audio...");
                        AbortIO(IOReq);   /* terminate command */
                        break;
                }

                puts("Playing...");

                Delay(25); /* this is where your real work goes */
        }

        /*
        ** Finish up by removing the IO request from the reply port.
        ** You can check for an error now.
        */
        error = WaitIO(IOReq);
        if (error) printf("I/O Error %d\n",error);
}
```

User Interface

CDTV Developer Notes #1
6/8/90

User Interface

The following section is a "Work-in-Progress" document.

In order to provide CDTV buyers with a consistent, easy-to-use interface, we have begun compiling a list of User Interface guidelines. This section represents the current thinking on this subject. Like other sections in this first issue of developer notes, this section is by no means complete.

We will be fleshing this out and adding to the list of User Interface Rules. We will also add more specific descriptions of start up sequence, audio control panel, and preferences.

We encourage your feedback on this material, particularly on areas that we may have missed, or areas where you believe there are problems. If you have questions, comments or ideas you can contact us on USENET, BIX, or by FAX. The numbers are:

USENET - rutgers/uunet!cbmvax!gswright
BIX - gswright
FAX - 603-878-4966

Guy Wright

CDTV - User Interface
Some notes on the structure of this document:

We have tried to organize this collection of ideas into logical areas. The table of contents shows the general structure of this document and how we have divided the various user interface issues into categories.

Following the Table of contents is a short introduction to User Interface standards.

Following the introduction is a list of what we felt are important UI guidelines, or rules, for developers designing applications for CDTV. The list is divided into categories using the same structure as the table of contents.
This list is by no means complete and we will be expanding it over the next few months.

Following the list of UI guidelines are short sections going into more detail on each of the areas outlined in the table of contents. These sections were put together for those people who would like to understand the thinking behind the UI guidelines.

The last section is reserved for ongoing discussions on general and specific UI issues.

****

User Interface Standards
Table of Contents

Purpose:

The purpose of these standards is to promote a uniform Look and Feel to most of the Baby applications.

I. Introduction

II. User Interface Rules

1. TV specific imagery

1.1 Issues related to 6 to 8 foot viewing distance and TV resolution.

2. I/O Issues

2.1 IR device
2.2 Other Input devices
2.3 Output devices

3. Screen Issues.

    3.1 Function
    3.2 Form

4. Operation of the machine, startup and audio only

    4.1 Startup sequence
    4.2 Welcome Disc

5. Application Specific Issues

    5.1 Start up
    5.2 Instructions
    5.3 Help function
    5.4 I/O concerns
    5.5 Error messages
    5.6 Key maps
    5.7 Exiting

6. New Ideas and Sources for Ideas

    6.1 Sources for ideas
    6.2 Starting point for new UI paradigms, and metaphors
    6.3 DB Formats for data to be presented
    6.4 Presentation Functional Areas
    6.5 Display Techniques

7. Information System Specific Issues

    7.1 Database Structures
    7.2 Atomicity of Data

8. Open End

    This space would be reserved for thoughts, comments, discussions, disagreements, etc.

****

Introduction to User Interface Standards

Alan Campagna - Theta Systems

Purpose:

The purpose of these standards is to promote a uniform Look and Feel to most of the Baby applications. What does "uniform look and feel" imply? As presented here it means that many of the Amiga

Intuition capabilities are disallowed as being too complex or un-intuitive to a new home user. It means that low level issues such as a small number of fonts and the colors to be used are strongly suggested. It means that "high level" functions,normally left to the application developer, such as control panels, list requesters, and scrolling functions are to be supplied by a toolkit (or descriptions provided prior to the toolkit being done.)

The effect of these standards will be to reduce developer options in choices of formats and modes of presentation in standard selection situations, thereby increasing the uniformity of look and feel to the user across many different applications. Note that this should have no effect on the developer's freedom of information presentation or content (ie, an animation or graphic, which is what the user selected, will be under complete control of the developer). Also note that by supplying the toolkit pieces, the standard fonts, and other elements of the UI standard, the level of effort for developers should be greatly reduced.

Characteristics

The User Interface for the baby should reflect the intended user population and environment; that is, it should be aimed at use on TVs in a home recreational setting by unsophisticated and often brand new computer users. Therefore it should have the following characteristics:

1) Ease of use.

2) Consistency of operation. Uniform look and operation across all similar applications.

3) Minimum documentation, or intuitive operation in any situation.

This document should be considered a work in progress. We invite your questions and comments. If you would like to become more involved in the creation of UI standards for CDTV then contact Alan Campagna or Guy Wright.

****

1. TV specific imagery

Issues related to 6 to 8 foot viewing distance and TV resolution.

Fonts. (some suggested fonts: Helvetica, Diamond, Times, but just about any fonts will work if they aren't too fancy in the first place. In completely un-scientific tests anything below 20 point type becomes difficult to read from more than ten feet. Also, broadcast television character generators almost always use anti-aliased fonts with a neutral

colored outline. Ideally, the outline color is halfway between the character color and the background color. Outlines, borders, and drop-shadows greatly improve readability. Pastel colored fonts work better than bright or primary colors. Off-white works better than pure white. Yellows, greys, and pale blues seem to work best.)
NOTE: We are commissioning a few sets of fonts in various point sizes. They will be designed specifically for the Baby and should be freely available to registered developers.

Colors. Colors should be subdued rather than bright (bright colors tend to 'bleed' on poorly adjusted TV sets.) If you are using a paint program like Deluxe Paint from Electronic Arts that have color values in a range from 0 to 15 re strongly recommend that no value go above 12 or 13. The only safe way to test colors and color combinations for display on an NTSC or PAL TV is with a waveform monitor. Maximum values 85% IRE. The best advice is to keep the color values below saturation and look at the results on a television set.

Nationality and user prefs selection. While many people will adjust the prefs (particularly nationality settings), most people will never touch them (if it ain't broke don't fix it). However, certain adjustments will have to be made by the user upon power up or after a loss of power. We will try to make the transition between preferences selection and normal use as seamless as possible.

Television sizes - Some people will be using small TV sets. While we might have colored borders around the edges of screen developers should work within a 'safe' area. We will be supplying specific details on this 'safe' area dimensions, in the meanwhile allow at least an inch on all borders (assuming you are using a standard Amiga monitor).

Television colors - Most people don't have their sets adjusted very well so don't rely on certain colors representing certain things. There is also the possibility that some people will be using black and white sets.

It should be remembered that Televisions are not the same as computer RGB monitors. Television is an interlaced, overscanned medium. What looks good on a monitor, may look terrible on a TV set in the home. It is always a good idea to view your screens on a TV set before you commit them to CD Disc. No matter how good your application is, if it doesn't look good on the home

TV set then the users will be disappointed.

****

2 - Input/Output Issues

2.1 IR device

As a positioning device any remote unit will be awkward to use for fine adjustments (even a mouse is not completely intuitive). For that reason we should not have gadgets for resizing, dragging, etc.

To avoid forcing the users to precisely position a pointer (and going along with some of the TV specific imagery rules) we felt it would be better to cycle through any list of selectable options.

We should establish a set definition of any key functions (such as Help or Return to main screen)

More thinking is required on how a user will use the IR device

for scrolling through lists, controlling text speed and size, etc.. There may come a time when an application requires some form of text entry and not everyone will own a keyboard.

We will be tackling these problems in the next few weeks and would appreciate any feedback or ideas.

2.2 Other Input Devices

We should start thinking about how people will use input devices other than the IR remote (keyboard, trackballs, joysticks, etc..)

One of the first thoughts is that wherever possible these other devices should behave or emulate the same functions as those on the IR remote. For example, the function keys on an optional keyboard should perform the same functions as those on the IR remote. Moving the trackball or joystick to the left or right should be the same as pressing the left or right arrow keys on the IR remote. Etc..

2.3 Output devices

One of the big problems we will have to deal with sooner or later is how people will interact with output devices. Right now all we have are questions.

- Printing. Since we are providing serial and parallel ports it is not unreasonable to assume that users will eventually

want to print things. Where do they get the printer drivers? Will we supply drivers on the Welcome disc or will drivers be included on application discs? How will a user select a printer driver? Will we supply a screen dump utility that will work for any screen?

- MIDI. Since we are providing a MIDI out port how will users use it? Will it be entirely application dependant or will there be occasions when a user will want to direct all audio output through a MIDI device?

- Video.

- Personal Memory Cards. This is a big issue that raises a number of questions. What sort of DOS will the user be using? What sorts of functions will they need to perform? (load, save, delete, copy, etc.)

- Disk drives. The personal memory card questions also apply to disk drives (hard or soft). What sort of DOS do we supply? What sorts of commands will users need? What sort of interface should we supply?

We have just begun to explore some of these issues. We haven't come up with all of the questions yet (let alone the answers) but we are working and thinking about all of these things.

****

3. Screen Issues.

3.1 Function

- NO GURU!!!
- Inform user when disk I/O or processing is going on. Like 'turning gears', 'Zzzzzz', or 'Working...' message. (Try for animations wherever possible.)
- Single window of activity... no window/task switch
- Support a standard way for user to navigate menu trees.
- No pointing.
- No double-clicking.
- No dragging.
- No typing. (All selections must be in a list or computable.)
- Where needed, use a type of split screen as a standard: if a second window is needed, allow for a function to put the window in one of the quadrants, (always a 1/4 or 1/9 screen sized window) : the border for the window is fixed by the called function, and the user cannot move the 'window' thus created.

- Always provide an Exit option

We are heading away from the traditional Workbench style interface. While it is infinitely easier than an MS-DOS interface it is still too complex and "computerish" for the living room environment. And if we can accomplish all of these things, there will be:

No manuals.

3.2 Form
- Single screen (to user)
- No Windows
- No window borders, because whole screen is window.
- No window gadgets. (No windows to close or resize.)
- No menus
- Ghosting (to indicate an option is not currently selectable)
- Uniform Exit icon

****

4 - Operation of the machine, startup and audio only

4.1 The following is a description of the CD-TV startup sequence, as outlined on May 18, 1990 in a meeting with Gail Wellington, Alan Campagna, David Rosen and Guy Wright.

User Interface Startup Sequence

There are five possible screens, or scenarios, that the user may be presented with upon starting up the CD-TV device. Which screen they first see will depend on certain conditions, such as cold boot, warm boot, CD-audio disk in drive, CD-TV disc in drive, etc.. There are also ways to move from one screen to another manually even if the conditions have not been met. For example, if the user wishes to change prefs even though they have been set before.

Conditions	Screen
Scenario 1	
Cold Boot Prefs not set in NVR (non-volatile RAM)	Set Preferences Screen (Prefs)

Scenario 2

Warm Boot/Reset Prefs set in NVR No Disc present	Insert Disc Prompt (ID)

Scenario 3

Warm Boot/Reset Prefs set in NVR CD-Audio or CD+G disk present	Audio Control Panel (ACP)

Scenario 4

Warm Boot/Reset Prefs set in NVR CD-TV disc present	CD-TV disc auto-boots

Scenario 5

Warm Boot/Reset Prefs set in NVR Unknown format or damaged disc present	Disk Error Message

The Insert Disc (ID) Screen

The ID screen would be similar to the "insert WB" graphic that Amiga users are presented with on start-up. Behind the ID prompt we might have the color cycling disc graphic. Later, if there is a CD-Audio or CD+G (with the graphics disabled) disc present, the user could return to this screen and see only the color cycling disc graphic without the "Insert Disc" prompt. Instead, the "Insert disc" graphic would change to a loaded disc, ready to be activated.

With no disc present the user has only one option from the ID screen.

1 - Exit to Prefs Screen

Of course, they could always insert a disc as instructed.

If there is a CD-Audio or CD+G disc present, but the user has exited the ACP back to the ID screen, they will have two slightly different options.

1 - Exit to Prefs Screen
2 - Select the "Activate Disc" icon (which, in this case, brings up the ACP screen)

Finally, if the user inserts an unrecognized format or damaged disk they will be presented with the Disc Error Message. This might be something like flashing the ID screen background colors on and off (unless we decide to

put the Error message on a separate screen).

The Set Preferences (Prefs) Screen

When the user turns on the machine for the first time, or if there has been a power failure then NVR will not have a copy of user specified preference settings. Any time the machine is activated and there is no copy of user specified prefs in NVR the first screen the user sees will be the Prefs screen. It is possible for the user to access this screen at other times in case they wish to change settings for some reason (daylight savings time changes, for example).

The Prefs screen will always give the user four options.

1 - Exit prefs (to ID screen)
2 - Adjust the Screen Centering
3 - Select Nationality (key maps)
4 - Set Clock

If there is a CD-TV application disc present (or any disc for that matter), but prefs are NOT set in NVR, the user would be presented with the Prefs Screen BEFORE the Application auto-boots.
Once the Prefs settings are changed and the user exits Prefs a copy of the user selected prefs will be written to NVR. If NVR Prefs is not set, and the user exits the prefs screen without setting anything, the default prefs will be copied to NVR.

*** Note: Should the user have to ***
*** change something before Prefs is ***
*** written to NVR? ***

If there is a CD-TV disc present when the user exits Prefs he will return to the ID screen for a moment while the application auto-boots. If there is a a CD-Audio or CD+G disc present when the user exits Prefs they will return to the ID screen where they can select the Activate Disc icon to get to the ACP.

The Audio Control Panel (ACP) Screen

The ACP is where the user will have options for playing back and controlling CD-Audio and CD+G discs. The specifics of the ACP will be supplied at a later date.

There will be an Exit icon on the main screen which will bring them up to the ID screen.

The Disc Error Message Screen

We should provide a way to signal the user that there is a problem with a disk. Either there is a hard error on the disk or they have inserted an unreadable format disc. This could be as simple as flashing the background color of the ID screen or as complex as a series of more specific error messages. Either way the system has to provide a way to signal the user that all is NOT well, rather than just locking up.

To the user there are only two main screens, ID and Prefs. The ACP will not appear to the user unless there is a disc present. If there is a CD-Audio or CD+G disc present the ACP will appear to the user as if it were an auto-booting, standardized application screen, (even though it will be in ROM.) The two main screen options that the user can access without inserting a disc are the Prefs screen and the Insert Disc (ID) screen.

```
*** Note: If there is a reason for a    ***
*** user to get to the ACP when there   ***
*** is no disc present then we could    ***
*** make ACP part of the three main     ***
*** screens.                            ***
```

Selecting Exit from ID will bring up Prefs. Selecting Exit from Prefs will bring up ID.
If there is a CD-TV, CD-Audio, or CD+G disc present then the Activate Disc icon should be the pre-selected default icon.

The actual screen displays for Prefs, ID, ACP, and Error Message will be worked out soon, and we will pass those along.

These few rules bring up some questions and implied rules for application developers.

1 - The system must be able to distinguish between a CD-TV disc and a CD-Audio or CD+G disc.
2 - All applications must be auto-booting.
3 - One of the very first things that an application should check for is a match on key-map settings in Prefs. If the application does not provide a language specific version matching the particular key-map selected in Prefs then the application must provide the user with an alternate choice or set of choices.
3a - For the sake of consistency CBM should provide certified developers with the code necessary to emulate the key-map selecting routine found in the Prefs screen.

This code should be easily modifiable to ghost key-maps that the application does not currently support.

4 - Is there a need for a user to get to the ACP if there is no CD-Audio or CD+G disc present?

5 - What sort of Error message should we provide in ROM? Should there be more than one type of error message?

6 - And a few dozen more.

If you have problems with any of this or ideas on improving the startup sequence then please let me know.

Guy Wright
May 23, 1990

4.2 Welcome Disc.

We are starting to put together ideas about what should be included on the Welcome Disc (the disc included with the machine). One item that should be included is a set of instructions on how to use the machine. Other items that have been suggested for inclusion on the Welcome Disc are application demos. These demos will perform two functions;

1. the demonstrate the capabilities of the machine
2. the serve as advertising for developers

More discussion and information about the Welcome Disc will be provided in the future.

****

5 - Application Specific Issues

Obviously, each developer will have their own ideas about how an application should perform, what it should do, and how to present it to the user. Some developers will completely take over the machine and do things their own way. Other developers will let the system and developer tools handle all aspects of presenting information to the users. In both cases we should still try and present the user with a consistent way of doing things.

5.1 Start up

> All applications should be auto-booting.
> All applications should check the key-map settings at start-up.

5.2 Instructions

If possible, all applications should try to provide any necessary

instructions on the disk itself. This can either be done at the start or through the Help function. It probably would be a good idea to provide a link to the Help function at all times.

5.3 Help Function

If possible (or necessary) applications should provide Help screens for the user. Since we will be designating one key on the IR remote as a Help function users will probably try and ask for Help no matter what the application.

If the application does not require Help screens then either bring the instruction screens back up or dis-activate the Help key function.

5.4 I/O Concerns

As stated elsewhere in this document, we will have to deal with I/O questions such as printers, MIDI, Personal Memory Cards, etc..

In some cases the application will have to handle I/O questions on their own, in other circumstances the system will handle them.

We realize that these will be very important considerations for many developers so we will try to provide information as quickly as we can.

5.5 Error Messages

Somehow we should come up with a standard set of error messages to inform users when something is wrong (Personal Memory Card Full, File Not Found, Memory Card Not Present, Printer Trouble, etc.

5.6 Key Maps

Ideally, all applications will be either language independent or provide versions of the program in all languages. Unfortunately, this is probably not going to be the case. Developers should be aware that this machine will be sold in many countries. If an application does not currently support the language selected in preferences the user must be presented with alternatives.

5.7 Exiting

It may be a minor issue in most cases because the machine will re-boot when a new disc is inserted (unless the program has specifically requested an override of the re-boot function for multi-disc applications). However, all applications should still try to exit cleanly. If any settings or preferences have been altered then they should be reset before exiting.

****

6 New Ideas, and Sources for ideas.

This section presents various short lists of topics which all are

associated with (a) the manner in which information is now currently presented on a computer screen, (b) the sources for ideas about new ways of presenting information, or (c) new ways which we might consider.

6.1 Sources for ideas

Look for ways in which information is currently (historically) presented (This was source of MFF).

- This is commensurate with form follows function (catalogue looks like catalogue, etc)
- Makes the presentation familiar to users.
- Also was the source of the desk top model.
- Separate indexed searches (structural searches) from browsing.

6.1.1 Current Computer GUI Paradigms

- Tabular data presentation;
- MFF scan of: word indices
  ordered "nodes" --
  node: = presentation script, with text, icon, buttons, pictures, animation

  - allows user sort on some keys --
    {fits the network "node" ideas}
- "Landscape: -- movement through a landscape of icons, or pictorial representation of the data
- Word Search/Indexed retrieval
- Browsing
- Menu --> control panel
- Game type
- "Mass Comp" building blocks and connectors
- Logo -- construction oriented
- Hypercard and hypertext

6.1.2 Sources of ideas, by product categories

- current Amiga products, and ideas.
- hypertext ideas
- MAC products, hypercard
- other existing products: current GUI SQL front ends.
- Object oriented graphics systems.
- Case
- Educational Stuff--Mac, Amiga

6.1.3 Sources of information, in the real world.

An examination of the physical world reveals a number of information sources that are easily tapped by individuals.

- Phone book; white pages (by name) Yellow pages; by category and name
- Encyclopedia
- Dictionary
- Catalog
- Magazine
- Library card catalog
- Rollodex cards
- Atlas
- Landscape scenes - physical movement
- TV Preview Guide
- TV Channels (switching)
- Microfilm
- Microfiche
- Television

6.2 Starting point for new GUI paradigms, and metaphors

6.2.1 Possible paradigms

- Bookshelf
- Carousel
- Map
- Store (shopping mall)
- Flipping channels

6.2.2 Possible metaphors

- Table (flat fixed files)
- Fiche
- Rollodex
- Outline
- Network
- Map(2D)
- Landscape
- Hypercard
- Split Screen, of various types

6.3 DB Formats for data to be presented.

- Record
- Network
- Ascii Text, with or without indexes
- Image data/Pixel/Bit Mapped
- Object (eg, CAD/CAM)
- Other graphics forms

6.4 Presentation Functional Areas

- Browse Paradigm: (eg, cockpit/aerial view/map/fiche window)

- Selection menu of fixed items:Icons/box selections/buttons/tools
- Network presentation: hypercard/hypertext; selection of next thing.
- Browse/select/order menu or tools, and mode selection.

6.5 Display Techniques

The physical information systems discussed in the previous section suggest a number of computerized display techniques.

A number of existing database display techniques will also be itemized here.

- Table
- Forms
- Hypertext
- Hypercard
- Outline
- Rollodex
- Network
- Fiche
- Map
- Scene with hot spots
- 3x5 Card with word search
- Free flowing text with word search
- TV "channels"
- VCR

****

7 Information System Specific Issues

Atomicity of data
Network versus unstructured data: user presentation capability
Presentation of selections from lists: visibility of indices, and presentation modes for bypass of keyboard type selection.

7.1 Database Structures

The previous sections discussed possible presentation paradigms, which are the external features of a database. No discussion about databases would be complete without a list of possible internal database formats.

- Record (fixed or variable) (with or without indexes)
- Network (e.g. Hypercard)
- Relational (interrelated flat files, with dictionary info)

- Ascii text (with or without indexes)

- Image (structured or bitmap)
- Object (e.g. CAD/CAM)
- Procedural (generates data dynamically)

7.2 Atomicity of Data

One of the broadest unanswered questions that must be answered before undertaking to design a new Visual Information System revolves around the "atomicity" of the data. This phrase that we have coined bears explaining.

A database with a high degree of atomicity is highly regular and lends itself to display using many different techniques.

A database with a low degree of atomicity is more "hard wired"; its display paradigm is determined at construction time rather than at runtime.

For example, an automobile parts list database may be constructed with either a high or low degree of atomicity. One constructed with a high degree of atomicity might include the part number, the part name, the part type, the assembly name, and the sub-assembly name for each and every part in the automobile. This database could then be displayed as a table, microfiche, outline, form, or any other format available today or in the future.

A parts list constructed with a low degree of atomicity may be modeled on an existing parts catalog, with exploded views of each assembly displayed graphically. This "hard wired" format does not implicitly allow multiple display formats.

This does not necessarily mean that a database with a low degree of atomicity is necessarily restricted to only one display paradigm, but is restricted to the paradigms allowed for at construction time. This is the way Hypercard works.

Some of us believe that providing powerful display paradigms which work with atomic data provides the most flexibility with the lowest cost for the data publishers, who construct the databases using the provided database tools.

Others believe that providing extreme flexibility in the presentation of the data, and, in fact, the ability to present each and every piece of data in the database differently will provide the nicest, most application specific presentations.

The final solution is probably to include aspects of both approaches. Solid and simple display paradigms should be available to the data publisher, while allowing the data publisher to tie together these displays with a more hard wired shell. In addition, it may be appropriate to provide a complete presentation language to the data publisher which would allow complete flexibility and an escape mechanism for future enhancements.

CES Report

CONSUMER ELECTRONICS SHOW
CHICAGO
2-5 JUNE 1990

Commodore presented the new CDTV player to the trade at the recent Consumer Electronics Show in Chicago. Six machines were set up in a special viewing area through which guests were escorted. The first machine was displayed in a typical living room setting connected to a TV. Here guest were briefed on the basic positioning of the machine as a family information and entertainment device.

Four more machines showed various types of applications demos. One the first station were the American Heritage Encyclopedic Dictionary and the Silver Palate Cookbook from Xiphias. In addition to introducing these applications, the ease of the user interface was demonstrated here.

The next station showed the World Vista Almanac demo from Applied Optical Media. With digitized Rand McNally maps, digitized images of various countries and their people, examples of digitized speech in many languages, and more, the breadth and depth of information accessible via CDTV was demonstrated.

The third station showed Fight of the Condor by Tiger Media, an interactive mystery with a radio-style sound track. This brought home to many visitors the true interactivity of the system.

Recreation demos were shown on the fourth system, Power Play from Arcana and Space Ace from ReadySoft.

On the last station in the demo area (nicknamed "the nursery" for obvious reasons), mediaMAXX showed off their sing-along disks and their Karakoe-style hardware add-on for CDTV. No great new talents were discovered, but everyone had a lot of fun trying out the system.

On Saturday night, Commodore hosted a well-attended press conference at the 410 Club. Speakers were Irving Gould, Chairman of the Board of Commodore and executive product manager for CDTV, and Nolan Busnell, General Manager of Commodore's new Consumer Interactive Products group.

The press releases from that gathering, a general press release and some of the press clippings follow.

Commodore
International Limited

Executive Offices
1200 Wilson Drive
West Chester, PA 19380
(215) 431-9100

FOR IMMEDIATE RELEASE

For further information:

Rogers & Associates
David Ellis
(213) 552-6922

Commodore International
David Rosen
(215) 344-3040

BOOTH 7043, McCormick North

COMMODORE PREVIEWS FIRST INTERACTIVE MULTIMEDIA HARDWARE SYSTEM AT SUMMER CES

CHICAGO, June 2, 1990 -- Commodore International unveiled its Commodore Dynamic Total Vision (CDTV) interactive multimedia player at Summer CES. The new player, which is a synthesis of Compact Disc and advanced computer technologies, represents a revolutionary advance in the development of interactive consumer electronics products and home entertainment.

Commodore's CDTV player is the first consumer-oriented product to combine the massive storage capabilities of the Compact Disc with the multitasking capabilities, audio, graphics, sound and processing power of an advanced personal computer into a single, simple to use, affordable unit.

"CDTV is the next logical step in the evolution of consumer electronics," said Nolan K. Bushnell, general manager of Commodore's recently formed Interactive Consumer Products division. "It provides capabilities far beyond any currently available entertainment or computer system, yet is remarkably simple to use. If you know how to change TV channels with a remote control, you can take full advantage of CDTV."

-more-

The player connects directly to a television set and home stereo unit to become an interactive entertainment, information and education center. A simple infrared remote device controls the player, allowing consumer access to a wide variety of programs. For example, CDTV technology makes it possible for consumers to access specific topics listed in reference materials such as encyclopedias, world atlases and almanacs at the touch of a button.

According to Bushnell, there is expected to be more than 100 titles available for the CDTV player at the time of product launch in the fall. In addition to various interactive resource materials, there is planned to be available a wide variety of instructional programs such as cooking, sports, foreign languages and special learning programs for children. Music, from rock to opera, will be enhanced with video graphics. Electronic games will be significantly more sophisticated in depth of play, sound and graphics than on any currently available video game or personal computer format.

The majority of discs will carry a retail price ranging between $30 and $100.

"We do not think of CDTV programs as software," said Bushnell. "Instead, the concept that drives CDTV is that of an electronic interactive, instantaneously accessible library. We will have fiction, non-fiction, reference and entertainment titles that provide consumers with a truly unique source of information and entertainment."

COMMODORE PREVIEWS FIRST
MULTIMEDIA HARDWARE SYSTEM
3-3-3-3-3

Commodore will offer a wide range of dedicated peripheral products as well. Planned peripherals include an infrared computer keyboard, infrared joysticks, a 3 1/2-inch floppy disk drive, a printer and a video genlocking card, which allow users to mix computer graphics with live video. The CDTV player is also compatible with conventional CD-Audio discs at 8-times oversampling and CD+G discs (Compact Disc audio with graphics).

The CDTV system of players, accessories and program applications is scheduled for launch in early fall by Commodore subsidiary companies operating throughout the world. CDTV's suggested retail price is expected to be under $1,000 (U.S.).

Commodore International, through its worldwide operations, is one of the world's leading producers of computer-based consumer and business products. Corporate headquarters are located at 1200 Wilson Drive, West Chester, Pennsylvania, 19380; (215) 431-9100.

# # #

(0503P)

Commodore
International Limited

Executive Offices
1200 Wilson Drive
West Chester, PA 19380
(215) 431-9100

FOR IMMEDIATE RELEASE

For further information:

Rogers & Associates
David Ellis
(213) 552-6922

Commodore International
David Rosen
(215) 344-3040

BOOTH 7043, McCormick North

COMMODORE CDTV PLAYER TARGETED TO ADULT CONSUMERS; NATIONAL DISTRIBUTION PLANNED

CHICAGO, June 2, 1990 -- Commodore's new interactive multimedia CDTV player addresses the growing consumer demand for interactive entertainment systems. The new player is the synthesis of advanced electronics and computer technology which provides consumers with the first truly interactive, integrated entertainment and information system.

Commodore Dynamic Total Vision (CDTV) brings advanced functionality and an enhanced audio and video experience without the operating complexity of computers or the dexterity required by game machines. CDTV will introduce consumers to a new way of learning and experiencing, bringing heightened sense of reality to conventional audio, video, text and graphics. Consumers can become active participants in music and books, sound will be enhanced by stunning graphics and visual images, and games will increase in complexity and challenge.

-more-

COMMODORE CDTV PLAYER
TARGETED TO ADULT CONSUMERS
2-2-2-2-2

Nolan Bushnell, general manager of Commodore's new Interactive Consumer Products division, identifies two groups of consumers for CDTV. The first group is enthusiasts who currently have advanced audio, video and computer systems in their homes.

The second group is educated adults who want to provide their families with interactive multimedia information, education and entertainment, but don't want to learn how to work with a conventional computer or buy a limited game machine.

"There is tremendous pent-up demand for a player with the capabilities of CDTV," said Bushnell. "Until now, consumers have had to deal with separate audio, video and computer systems which were unrelated to each other," adds Bushnell. CDTV integrates consumer electronics technology with the computer into a single, easy-to-use system, which adds new dimensions to the video and audio experience.

Bushnell anticipates that market acceptance of CDTV will follow a pattern similar to that of the VCR. Commodore plans to launch the product internationally. While precise distribution plans have yet to be finalized, the company plans to market the product in the United States through high-end audio/video specialty retailers, upscale department stores and selected retail chains.

-more-

COMMODORE CDTV PLAYER
TARGETED TO ADULT CONSUMERS
3-3-3-3-3

"Our retail distribution will reflect the specific needs of our target audience," said Bushnell. "Initial CDTV buyers are seen as sophisticated consumers who typically spend a great deal of time researching and gathering information before making purchases. The retailers selected to introduce this product must know not only how to sell and explain new technology, but provide the high levels of service required to meet the needs of these discerning consumers."

Commodore plans to have a library of more than 100 applications for CDTV available when the product launches in fall, with 200 titles available for Christmas 1990. Titles for CDTV will be inclusive of entertainment, information and education. Planned programs will consist of reference materials, interactive learning programs for children, games and fiction, as well as exciting instructional programs that include leisure activities such as sports and cooking, business skills and self-improvement. This range suggests a fraction of the potential topics that can be available on CDTV.

Commodore International Limited, through its worldwide operations, is one of the world's leading producers of consumer and business products.

# # #

(0509P/1)

Commodore
International Limited

Executive Offices
1200 Wilson Drive
West Chester, PA 19380
(215) 431-9100

COMMODORE ANNOUNCES THE CDTV PLAYER

Home Information/Education/Entertainment System

For New Generation of Interactive, Multimedia Applications

WEST CHESTER, PA., JUNE 4, 1990 -- Commodore International Limited (NYSE:CBU) announced the CDTV Player at the Consumer Electronics Show in Chicago on Saturday, June 2.

The CDTV Player, which easily connects to any television, stereo system or video monitor, operates a new generation of interactive, multimedia applications on compact disc for education, information, reference and entertainment. The attractively-styled unit is easily operated through a ten-key infrared remote control.

The CDTV player will be marketed by Commodore's worldwide consumer sales force, and is planned to be sold later this year through better consumer electronics/audio video departments and specialty stores. It is expected to retail for under $1,000.

Nolan Bushnell, General Manager of Consumer Interactive Products for Commodore, said: "It's what everybody has been waiting for ... a product that lets you change your home television set from a passive to an active medium. This unit can turn any couch potato into an active participant in their own education and entertainment."

By utilizing the CDTV's advanced capabilities and expansive memory (each disc has more memory than 700 computer diskettes), the CDTV applications are expected to open new dimensions in learning and experience through ease of use, true interactivity, and an innovative mixture of words, graphics, motion, sound and videographic detail. Lucasfilm, the Virgin Publishing division of Virgin Records, Cinemaware and other third-party developers are scheduled to ship over 100 CDTV application discs this year. The discs are currently expected to retail around the $25-$60 range.

In addition to various interactive resource materials, a wide variety of instructional programs are planned, in cooking, sports, foreign languages and special learning programs for children. Music, from rock to opera, will be enhanced with video graphics, while electronic games will offer significantly more sophisticated depth of play, sound

and graphics. The CDTV Player will also run existing compact disks featuring music, and music and graphics (CD+G).

Key to the CDTV is an innovative blend of CD-ROM technology, the Motorola 68000 microprocessor and Commodore's proprietary multimedia technology currently used in its line of Amiga personal computers. The Amiga is the only computer with out-of-the-box capability to multitask and manipulate video, graphics, animation, music and text.

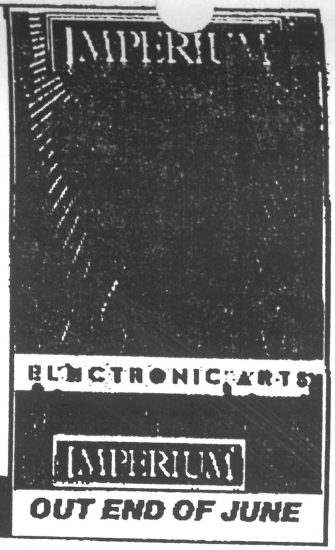
The CDTV Player can be expanded to become a personal computer simply by plugging in optional extras such as a keyboard, with an infrared option, and floppy and hard disk drives. A peripheral that will run CDTV discs on Commodore's Amiga computers is planned.

Commodore has a long history of innovation in the consumer electronics and personal computer business. The Company popularized the calculator in the early 1970s, developed one of the first personal business computers in the late 1970s, and introduced one of the first home computers in the early 1980s.

"This is an important day for Commodore," said Chairman and Chief Executive Officer Irving Gould. "The CDTV Player showcases the vast potential of our Amiga multimedia technology and represents a new product category and market for home information, education and entertainment products."

CTW

4.6.90 COMPUTER TRADE WEEKLY ISSUE 289



CBM storms out with £699 CDTV

Commodore this week finally unveils its CD Amiga — a system which it feels could become the standard in a colossal new market for "complete home entertainment systems".

Called the CDTV, the machine (first details being broken by *CTW* way back on January 15th) boasts a black central unit which is meant to be able to sit equally comfortably in major chains' VCR/TV/hi-fi sales areas as well as computer departments.

Availability: After going on show at Chicago CES from Saturday (June 2nd), harassed Commodore staff around the world can finally stop denying the CD Amiga's existence — but there's still no official word on the key area of availability.

A US insider told *CTW* last week, however, that the system should become available in August. UK availability isn't expected to be until late September/early October — although it will be shown here for the first time at EMAP's Computer Entertainment Show at Earl's Court on September 13th.

Price: Again, nothing official here in case any unforeseen last minute adjustments are required. *CTW* can reveal again, however, that a US price of around $900 is likely and in the UK it is already pretty much set at £699.

Tech spec: A high-end CD audio unit (with the latest CD+G option), an infra red handset, in-built Amiga PCB with 1Mb of memory and fully-compatible with all existing A500 software if a 3.5 inch disk drive is added.

Each CD can handle 660Mb, the equivalent to 720 3.5 inch disks. There is also "a form of storage" using 64K smart cards which slot into the front.

Extras: Users will be able to buy an infra red keyboard, trackerball or joysticks for use with the CDTV. These will work via the separate transmission device called 'the brick' — which can also turn standard joysticks into infra red ones.

Development: Commodore has been working on the system for 18 months. It is also known that there has been some Japanese involvement on the project.

Software: Some 40 'titles' — *not* necessarily games — are expected to be available as soon as the system is launched. Dedicated games firms are already working on new software or transferring old games from disk to CD.

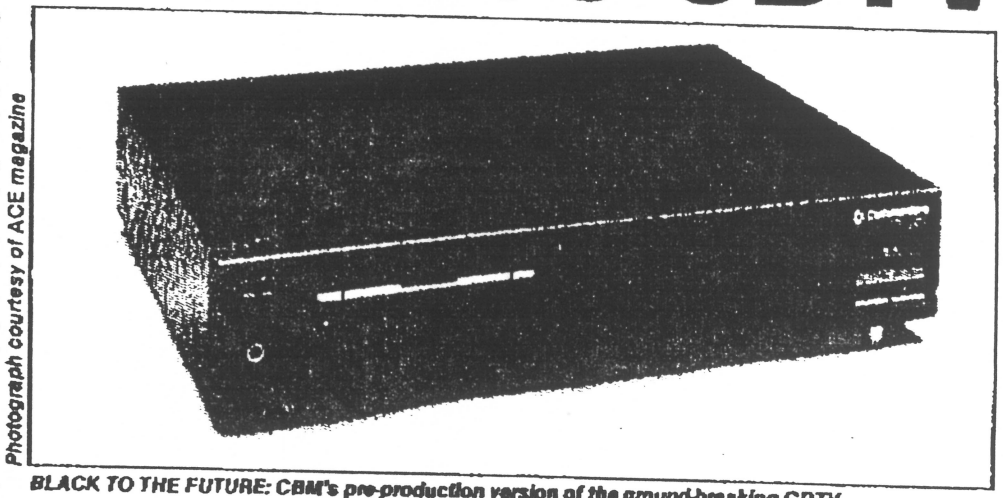
Marketing: Clearly the firm sees the CDTV as a genuinely mass market product — particularly in Europe. The name of the thing itself has been deliberately chosen so that consumers may more easily understand that it is "an extension of TV — making a TV interactive instead of passive".

There are a whole range of possible markets, ranging from education to medical to storage of government records to audio to computer games. The major market will, however, be the home and UK chains have apparently already indicated that — if properly handled — they think the CDTV could be enormous over the next three years.

Commodore's UK boss Steve Franklin told *CTW* last week that he had been guaranteed enough units this side of Christmas to warrant a fair-sized marketing burst — including TV ads.

"The whole thing isn't about the hardware, it's all about how you market it. It's like when videos first came out — your marketing has to tell people what the thing can do. It's not just a games machine or a CD player, it'a lot more," he explained.

"Obviously there's a lot of pressure on us to get it right. We could possibly be setting the standard for the future."



Photograph courtesy of ACE magazine

***BLACK TO THE FUTURE:** CBM's pre-production version of the ground-breaking CDTV*

HOME INTERACTIVE SYSTEM PROMISED FOR FALL: Biggest surprise of Consumer Electronics Show -- and a secret so well-kept few potential competitors knew about it even after it was announced -- was CDTV, CD-based consumer interactive audiovisual system promised for fall by Commodore's new Worldwide Interactive Consumer Products Div., headed by Nolan Bushnell, originator of the videogame.

CDTV doesn't stand for exactly what you might think, but it might as well. Officially, it's abbreviation for "Commodore Dynamic Total Vision" (TVD June 4 p13), system obviously designed to steal march on Philips-backed CD-Interactive (CD-I) that isn't compatible with CDTV. Commodore said it will have system on market in fall at under $1,000, aims for $500 price next year. News conference was told there would be "way over 100" audiovideo program discs available at launch, 200 by Christmas, but company spokesman at exhibit said goal was 25 multimedia programs at start, 75-100 by Christmas, leading to conclusion that difference is represented by CD+Graphics programs, which system also will display.

Bushnell, 46, who was responsible for starting first videogame as head of Atari, told us Mitsubishi would make 100,000 CDTV hardware systems on OEM basis for launch this year. He said Commodore soon would negotiate to license other consumer electronics manufacturers to market system. He hinted Mitsubishi also was interested in marketing CDTV under own name, but officials of Mitsubishi Electronics Sales America (MESA) at show were unaware of system or of deal. "We have an obligation to license others," Commodore Chmn. Irving Gould said.

Chicago CDTV news conference was "an announcement, not an introduction," Commodore said, with formal details to come within several weeks -- including names of software developers. System, based on Amiga computer chip set, will be attachable to any TV in world, Commodore said (it will have NTSC, PAL and RHB outputs) and CDs for system will vary from cookbooks, atlases and encyclopaedias to sophisticated games and even crossword puzzles with audio and video cues, most CD programs listing at $30-$100. It also will play audio CDs and display CD+Graphics. Hardware is convertible to Amiga 500 computer with addition of optional keyboard and other peripherals. TV-like remote control is used as mouse for interactive programs.

"The beginning of the next generation of consumer electronics" was what Gould called CDTV. Bushnell was even more effusive, predicting CDTV could replace book as reference tool and update American education system. CDTV doesn't provide whole-screen full motion." Commodore officials said it can display half-screen full motion or 15 frames per sec. full-screen. However, Bushnell said CDTV was "open architecture system," and when International Standards Organization (ISO) approves standard involving full motion, Commodore will offer update within 3 months.

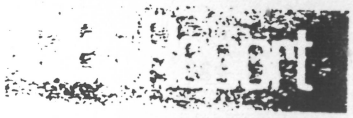
Marketing and distribution plans haven't been set, executives said, pointing out that talks with dealers are in preliminary stage (Commodore also introduced consumer version of Amiga on consumer market at list price of $799 including $250 worth of software). Although no list of software suppliers was released, representatives of some were at news conference and at Commodore booth, firms including Applied Optical Media, Cinemaware, Tiger Media, Xiphias. At news conference, some software developers expressed impatience at slowness of standardization efforts and delays in bringing CD-I system to market. Bushnell said Philips "is still saddled with cost restraints and problems [on CD-I] that we've solved." CD-I is 4-year-old technology.

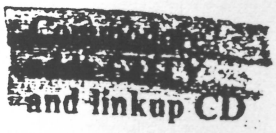
Philips didn't agree with Bushnell that CDTV was "one of the momentous events of the decade." Philips Interactive Media Systems Mktg. & Sales Vp Gerald Calabrese learned of CDTV at CES and hadn't yet seen it when we talked with him. He warned that no standards exist for CD-based interactive video systems and noted that CD-I has been proposed to ISO as standard. He doubted that CDTV, as proprietary system, could be made compatible with CD-I.

Philips plans mid-1991 consumer launch of CD-I as full-motion full-screen system "from day one," Calabrese told us. Standardization is important "to avoid creating confusion in the marketplace," he said: "Philips has been very careful in following the same procedures we did in CD audio -- any CD will play on any player anywhere in the world. A proprietary product would require proprietary software, which won't play on CD-I. We would be on the market today, but we want a complete product with a full-screen processor and software available. Philips believes world standards are important for success in any consumer product." Philips in past has said it's aiming at under-$1,000 price for hardware.

Although CDTV will play games, it's being aimed largely at adult market in homes, but with schools also primary target, Bushnell said. "My first goal is to fundamentally change the nature of education in America," videogame pioneer said. "It's my way of saying I'm sorry."

Home Furnishing Daily
4 June 1990





By Mark Seavy

CHICAGO—Commodore International, hailing what it said is the dawning of a new generation of electronics, Saturday unveiled its Commodore Dynamic Total Vision (CDTV) player.

The CDTV, which is scheduled to be shipped to retailers in the fall, is similar in function and appearence to a conventional CD player but is married to a Commodore Amiga 500 computer. It will have a suggested retail price under $1,000.

"We believe that CDTV ushers in a new era of communications entering the new vocabulary of images, songs, and capabilities which will change forever how we communicate, how we learn, and how we will be entertained all at a very affordable price," said Irving Gould, Commodore's executive product manager and chairman of the board of directors.

"In the past, consumer electronics consisted of several parallel industries. The worlds of audio, video and computing were separate with distinct functions and technologies. Each aspect represented an attempt to replicate part of the human experience. The introduction of CDTV brings us one step closer to cohesively unifying these communications media through their interactive integration on the Amiga platform."

Initial shipments to high-end retail and specialty stores in the United States may run between 35,000 and 50,000 units with an expected worldwide roll out of 100,000, said Nolan K. Bushnell, the founder of Atari Corp. and now general manager of the Consumer Interactive Products division of the West Chester, Pa.-based company.

Bushnell said the product will be targeted for consumers with audio, video and computer systems in their homes as well as adults seeking to bring information, education and entertainment to their families.

He said an offshoot of the education market could be school systems nationwide. There is a potential for sales of two million units to schools within two years, he said.

FINANCIAL TIMES WEDNESDAY JUNE 6 1990

London

Compact disc goes on TV

A HOME computer system which exploits compact disc technology to give sound and colour graphics will be on sale this September from PC specialist Commodore.

The CDTV player, which will sell for £999, looks similar to the compact disc player which graces many a home. But inside the box, alongside the disc player itself, is a Commodore Amiga computer.

When plugged into the television set the machine can be used to play games or, because it has CD quality sound, for learning languages. It can also be used just for listening to music.

To operate the machine the technophobic can use an infra-red handset, similar to that used to change the programmes on the television, or an infra-red keyboard, mouse-type device or joystick.

The CDTV player uses discs each of which has the equivalent capacity of 700 floppy discs or 250,000 pages of text.

Wall Street Journal
5 June 1990

Commodore Unveils System Linking CD, Computer Functions

By a Wall Street Journal *Staff Reporter*

WEST CHESTER, Pa.—Commodore International Ltd. introduced a system that combines the capabilities of a personal computer and compact disk player.

The Commodore system, called the CDTV player, consists of an Amiga personal computer—without keyboard or monitor—and a CD player. The system is hooked up to a television to display video and text.

The CDTV Player makes the personal computer maker an early entrant into a market that many computer companies consider to be vast. The companies think that the home market for computers is finally about to pick up with the introduction of multimedia machines. For instance, they expect people to buy encyclopedias on computer disks with video and audio capabilities, or instructional videos that they can interact with through the computer.

Commodore has also broken through a price barrier. Its CDTV Player, whose shipment date wasn't specified, will cost less than $1,000. Application disks will cost just $25 to $60.

Still, the market for such systems is very young and will take time to develop. The various companies involved haven't agreed on what technological standards they will follow, and it usually takes the market quite some time to sort out those issues. It's also not clear how people will use these systems, and whether they will want reference books or just more games. In any case, it may take a couple of years for companies to produce enough application disks to make these systems attractive.

"Commodore has a history of producing great hardware, but they don't have enough software available," said Greg Drahuschak, a securities analyst at Wheat First/Butcher & Singer. "That is where Commodore has always fallen short."

COMMODORE PICKS UP THE GAUNTLET TO DRIVE CD-ROM INTO THE CONSUMER MARKET

Picking up where Headstart Technologies (O&MR, December 1989, p. 1) left off and going one step further, Commodore International Ltd. has taken the lead in attempts to create a consumer market for CD-ROM. At summer CES, the company unveiled what it calls CDTV (Commodore Dynamic Total Vision), an interactive CD-ROM player aimed squarely at home users. Scheduled to ship in September, it gives Commodore nearly a year's lead on what promises to be its major rival, CD-I, which should be introduced by Philips and a host of others in the summer of 1991.

Commodore's effort was spearheaded by Nolan Bushnell, the founder of Atari, Commodore's arch rival in the low-end computer wars of the early 1980s. Bushnell is now general manager of Commodore's Interactive Products Division.

The $895 unit, which is based on the Amiga 500 computer, includes a megabyte of RAM and is contained in a VCR-like box that connects directly to either a television set or an RGB monitor. To control the programming, users are offered an option of a remote control or an infrared mouse. A keyboard is optional. The player also has a modem and S-VHS jacks.

Based on the international ISO 9660 standard for CD-ROM, CDTV will be able to play games, reference works and educational software developed for CD-ROM as well as audio CDs and CD+G. One catch--it can only use software developed with the company's new Amigavision authoring system (O&MR, May 1990, p. 10). But that may not be a major drawback and could have some advantages. "I understand it will require just some minor tweaking" to port existing CD-ROM software to Amigavision, says one software developer who plans to support the format. Adds Commodore Chief Engineer Don Gilbreath, "In the future, developers will design video and databases to work across all platforms."

He is probably right. "While we haven't announced anything, there is a lot of hardware out there. We have talked to Commodore and if there is going to be a consumer CD-ROM player, we will support it," says Kirk Green, director of public relations at Sierra On-Line, a leading publisher of PC entertainment software.

On the upside, Commodore is the first company to come out with compatible multimedia systems aimed both at the professional and the consumer markets. Software developed on its new Amiga 3000 line of computers can be played on CDTV. CDTV can serve as a low-cost industrial playback unit as well. Clearly, the cross-over potential is enormous.

In any case, PC entertainment software publishers are itching for consumer-priced CD-ROM to emerge to help them better compete with Nintendo and the low-end video game market. To help seed the market, this fall Sierra On-Line plans to bundle a CD-ROM drive, a PC sound add-in board and two entertainment programs--"Mixed Up Mother Goose" and "A Kings Quest 5"--and sell it for around $700. The same approach worked for Sierra two years ago when it wanted to improve the quality of PC sound and bundled games with a sound board, Green notes. Several other PC entertainment software publishers reportedly will release CD-ROMs this fall. (Commodore International, 1200 Wilson Drive, West Chester, PA 19380, 215/431-9100 Sierra On-Line, PO Box 485, Coarsegold, CA 93614, 209/683-4468.)

....WHILE JAPAN POSITIONS ITSELF TO PLAY A MAJOR ROLE IN THE CD-ROM MARKET

As successive U.S. companies take runs at establishing the CD-ROM market while simultaneously scrambling to develop the next generation of technology, the Japanese government and related institutions have taken a systematic approach to nurturing the market. As New Technology Week reports, the Ministry of Finance has put the financial statements of 1,800 companies on CD-ROM; the Polymer Materials Center provides a database on 65 companies, 3,000 catalogs and 2,000 documents in English on CD-ROM; and the Japan Electronic Publishing Association recently issued a Japanese language CD-ROM format based on international standards. The Japanese Ministry of International Trade and Industry (MITI) and 22 industry organizations are testing the standards. The theory--build volume for CD-ROM players quickly. The payoff could come soon. One insider says Japanese companies could ship CD-ROM players selling for around $200 as early as the fourth quarter of this year.